AI in physics: selected studies in classical and quantum mechanics

JACK GRIFFITHS

AI in Physics

Selected studies in classical and quantum physics

Jack Griffiths

AI in Physics

Selected studies in classical
and quantum physics

Jack Griffiths

A thesis presented for the degree of

Doctor of Philosophy

Department of Physics

University of Durham

United Kingdom

August 2024

A Ph.D. is often a solitary project, but there have been many people who have supported my work whilst I have been working towards this thesis.

Thank you to my supervisors, Simon and Steven, for your help, and wisdom, throughout my Ph.D.

Thank you to Anthony, Gareth, Sam and Miloš for being incredible friends, and helping me through some of the most difficult periods of my life.

Thank you to Tom and Clarissa for hosting me in Innsbruck, and making some excellent memories skiing, with the Baby Jesus in Verona and Einaudi in Milan.

Дякую, Олександро, за дружбу, меми та багато фотографій Джековича. До зустрічі в Україні цього року!

Thank you to my mam, Sonia, my sisters Janette, Elaine, and Gwyneth, my brother Paul, and my brother-in-law Paul for your support in everything that I do.

Thank you to Alice and Evelyn for your lifetime of love and friendship.

And thank you to my late father, Brian, for everything that you have taught me. I love you, and I miss you.
This thesis is dedicated to your memory.

Abstract

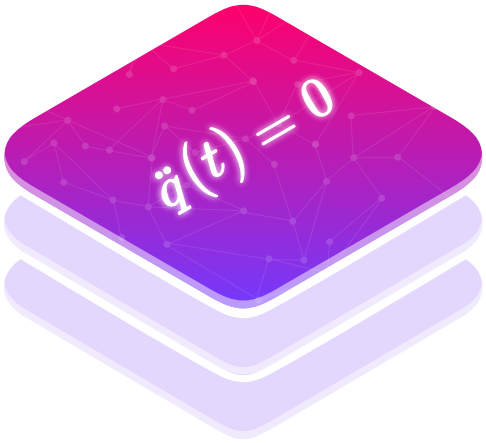
We explore three questions at the intersection of physics and artificial intelligence (AI): Can we use techniques in AI to solve physics-based initial value problems? Using this framework, can we solve Sturm–Liouville problems such as the general Legendre equation and the time-independent Schrödinger equation? And can we use AI to predict thermodynamic parameters of a Bose-condensed cloud of atoms to solve a current bottleneck in quantum fluids research?

Solving initial value problems using AI has yet to be implemented in the strictest sense. Many approaches in the literature require sparse data from the solution of the initial value problem in order to interpolate between these data points. We present the *neural initial value problem*—a framework which can approximate solutions to a range of dynamical systems without *a priori* knowledge of the solution. We first consider the most minimally conceivable neural network to solve an initial value problem describing free particle dynamics. We then extend this framework to solve a range of non-linear, coupled, and chaotic problems including the classical pendulum and the Hénon–Heiles system. We introduce *probabilistic activation functions*—it is our observation that these are required to find solutions of initial value problems. We also introduce *coupled neural networks* to solve systems of differential equations. These frameworks further allow us to solve eigenvalue problems such as the time-independent Schrödinger equation in a harmonic trap or the Legendre equation.

Numerical schemes for finite-temperature Bose gases require knowledge of the chemical potential and temperature, which are imprecisely and destructively determined in experiments. We demonstrate a proof-of principle AI that can predict these parameters given only a two-dimensional atomic density profile. The model produces accurate predictions for harmonically trapped condensates and can also produce accurate predictions on toroidally trapped condensates despite never being trained on such a trap. The model can also predict the values of these parameters during the thermalisation procedure. Since predictions happen in fractions of a second, this model could be used for real-time analysis of these parameters in experiments.

Neural initial value problem
Part III, Chapter 4

Approximating solutions of initial value problems in classical mechanics with machine learning techniques.

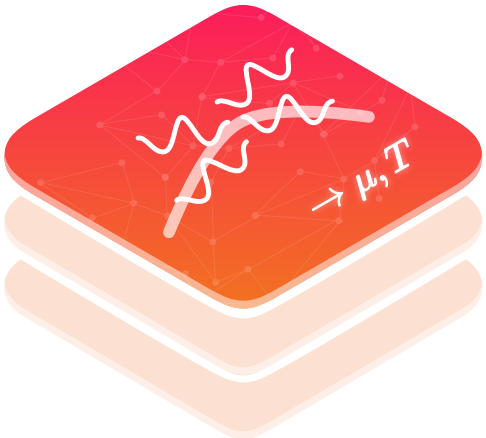


Neural Sturm–Liouville problem
Part III, Chapter 5

Approximating eigenvectors and eigenvalues of Sturm–Liouville problems with machine learning techniques.

Machine learning thermodynamical parameters of quantum fluids
Part III, Chapter 6

Predicting the chemical potential and temperature of a Bose gas using AI.



Data availability

All code and their documentation is available on GitHub:

- *Neural initial value problem solver*: the Python and PyTorch code to reproduce our results in part III, chapter 4. Examples are given and interacted with via the command line.
  URL: `https://github.com/0jg/neural-ivp`.
- *Neural eigenvalue problems*: the Python and PyTorch code to reproduce our results in part III, chapter 5.
  URL: `https://github.com/0jg/neural-eigenvalue`.
- *Stochastic Gross-Pitaevskii equation in Rust*: the Rust code that we used to generate training data for part III, chapter 6. It supports both harmonic and toroidal traps, and is mostly interacted with via the command line.
  URL: `https://github.com/0jg/sgpe-rs`. Also available as a Rust crate at `https://crates.io/crates/sgpe`.
- *Chemical potential and temperature prediction model*: the Python and PyTorch code to reproduce our results in part III, chapter 6.
  URL: `https://github.com/0jg/mu-t-predictor`.

Previous publications

This thesis contains work which has previously appeared in publication or is in preparation to be published:

- J. Griffiths, S.A. Wrathmall, and S.A. Gardiner. *Solving physics-based initial value problems with unsupervised machine learning.* arXiv preprint arXiv:2407.18320 (2024) [GWG24].
- J. Griffiths, S.A. Wrathmall, and S.A. Gardiner. *Temperature and chemical potential characterisation of Bose–Einstein condensates with artificial intelligence. In preparation.*

Contents

List of Figures

Part I

Overview

1

ARTIFICIAL INTELLIGENCE IN PHYSICS

Artificial intelligence (AI) is the ability of a computer to mimic human intelligence, encompassing logical reasoning, natural language processing, and learning capabilities. The latter concerns the field of machine learning: can a computer learn the intricate patterns and structures in data, beyond what a human could learn? The growth of AI and machine learning applications has been made possible by an increase in GPU performance and availability, allowing bigger models to be trained. Machines learn through a class of algorithms known as deep learning—an interplay of matrix algebra, multivariate calculus and statistics. We introduce these mathematics of deep learning in §§ 2.2–2.8. The relationship between AI, machine learning, and deep learning is shown in Figure 1.1.

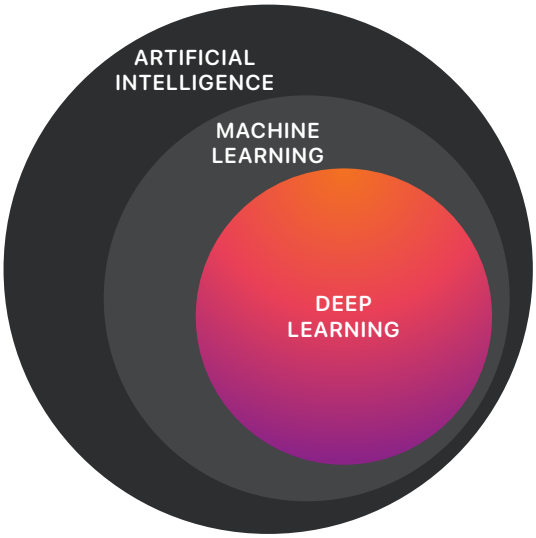


Figure 1.1: Machine learning is a subset of AI. Deep learning covers a general class of algorithms for machine learning tasks. [Figure adapted from various authors.]

1.1 Intersection of physics and artificial intelligence

The increasing complexity of the simulation of physical problems and the growing demand for accurate simulations has always motivated the development of new computational methods; the intersection of physics and AI represents a growing and significant field of interest in the scientific community.

AI can be augmented with physical constraints, equations, data, or other physical principles (such as the Rayleigh–Ritz variational method or Hamilton's principle).[1] In part III, chapter 4, we learn representations of the solutions of initial value problems using machine learning, guided by Hamilton's principle. In part III, chapter 5, we learn eigenfunctions and eigenvalues of Sturm–Liouville problems such as the

[1] In the literature, using machine learning to solve differential equations is sometimes referred to as a 'physics-informed neural network' (as introduced in, e.g., [RPK19]). Using physics priors in any AI or machine learning context—not just to solve differential equations—is referred to as 'physics-informed machine learning'. Some implementations of AI in physics fall outside of the usual definition of machine learning. Because of these inconsistencies in this terminology, we avoid their usage in this thesis, and instead prefer to describe the entire field as *physics-informed AI* and describe specific implementations (such as solving differential equations with machine learning) explicitly.

general Legendre equation and the time-independent Schrödinger equation. In part III, chapter 6, we use AI to predict the chemical potential and temperature of a Bose gas by using data from simulations.

Machines can learn from existing data from experiments or simulations to make predictions or decisions—this is *supervised* learning. Machines can also learn without data so long as the problem is constrained sufficiently well—this is *unsupervised* learning. Predictions from machine learning models are compared to the actual outcomes using a cost metric (also called a loss or objective function), which measures the accuracy of the predictions. An optimisation algorithm adjusts the parameterisation of the model to minimise this cost metric, iteratively improving the model's performance—this is referred to as *training*. Once trained, the model is often evaluated using separate data to ensure it can generalise well to new, unseen data. Once tested, the model is ready for rapid inference, enabling it to quickly make (hopefully) accurate predictions or decisions based on new input data. When learning solutions to differential equations (in chapters 4 and 5), we do not seek generalisation capabilities of our models; we instead verify that the solutions are physically acceptable by evaluating, for example, the numerical action.

Physics priors can be incorporated into AI in several ways:

- **Model architecture:** The underlying architecture of the AI can be designed to respect physical laws. For instance, Ling, Kurzawski, and Templeton [LKT16] developed a deep neural network to predict turbulent fluid flows, incorporating an auxiliary network to ensure invariance to Galilean transformations, which are expected in classical fluid dynamics. In part III, chapter 4, we use coupled models to solve systems of differential equations. When solving the time-independent Schrödinger equation in part III, chapter 5, we simultaneously obtain the eigenstates and eigenenergies in a dual optimisation scheme.
- **Cost function:** Physical principles can be embedded in the cost function, guiding the optimisation process to produce physically meaningful results. We discuss this further in § 1.2.

- **Training data:** Physical data may be present in the training set, sourced from experimental results or simulations [GDY19; Cra+20]. Training data can also be augmented [LeC+95b] using transformations like translations, scalings, or rotations to reflect the known symmetries of the system, thereby avoiding the need to generate new data.

1.2 Machine learning representations of initial value and Sturm–Liouville problems

Machine learning representations of initial value problems are found by introducing physics priors in the cost function and in principle by feeding the model with some (possibly sparse) training data. The cost function may consist of a weighted combination of: i) the differential equation (or other equations which constrain the solution, such as whether the solution should be divergence free [WY23], or any other mathematical description of the data [RPK19]), ii) the initial conditions (where appropriate), iii) the boundary conditions (where appropriate), and iv) sparse data sampled from throughout the problem domain (the training data). Other problem-specific conditions may also be included in the cost function. Xu, *et al.* [Xu+20], solve Fokker–Planck equations by additionally constraining the normalisation of the solutions (which are probability distributions) in the cost function. The training data (previously observed or simulated solutions of the differential equation) may be sparse, as this approach effectively interpolates between data points by enforcing the constraint given by the differential equation.

Our experiments and the research of Vapnik [Vap95; Vap82] emphasise the importance of choosing the simplest cost function possible to avoid overfitting neural networks or to avoid failure of training altogether. One may know the differential equation, initial conditions, Hamiltonian [GDY19], and Lagrangian [Cra+20], but one should not, in general, include all of these terms in the cost function.

We introduce a framework to solve initial value problems in the strictest sense, that is without any constraints which are not needed mathematically. This is an important

divergence from the literature: we do not include any extraneous regularisation terms as is common in typical physics-informed neural networks [LLF98; Kar+21; HJE18; SS18; Lu+21; Xu+20]. We call this the *neural initial value problem*. The neural initial value problem is an implementation of Hamilton's principle through machine learning methods; for all dynamical problems we consider, the only constraints are the underlying differential equation(s) and the associated initial conditions. There is no training data. Our discussion starts with the most minimally conceivable neural network to describe free particle dynamics—this allows us to set notation and introduce our neural network representation of initial value problems. We consider linear approximations of a particle in a gravitational field and in a harmonic oscillator. We briefly discuss observed deviations from the theoretical results of the harmonic oscillator. Guided by the need to increase the representational capacity of our neural networks, we introduce deep learning techniques to find solutions of the classical pendulum. We introduce *coupled neural networks*—alongside associated optimisation routines—using the Hénon–Heiles system [HH64] (akin to three-body dynamics [BGM98]) as an example. We finally make comments relating the machine learning trajectories back to Hamilton's principle—our solutions are eventually consistent with the principle of stationary action, despite some peculiarities in how the action evolves during training.

Figure 1.2 shows a black box model for the neural initial value problem. A neural network is akin to a function machine: by solving an optimisation problem, we learn the mapping between input data and output data in a way that minimises the cost function. The inner workings of the neural network are often regarded as a black box that is seldom opened (even by machine learning practitioners). A secondary objective of this thesis is to elucidate how machines learn and to open the black box.

We extend the neural initial value problem to consider Sturm–Liouville problems which additionally require the optimisation of an eigenvalue. We introduce the *neural Sturm–Liouville problem* to find the eigenstates and eigenenergies of the time-independent Schrödinger equation. By allowing for the free optimisation of the

eigenenergy as a parameter in our machine learning model, alongside optimisation of the eigenstates using the neural initial value problem framework, we are able to quickly and reliably find physically acceptable states of a particle in a quantum harmonic oscillator. The implementation successfully obeys the quantisation of energy in the harmonic oscillator trap. We also use the neural Sturm–Liouville problem to find the *associated Legendre polynomials* and, in turn, the spherical harmonics.

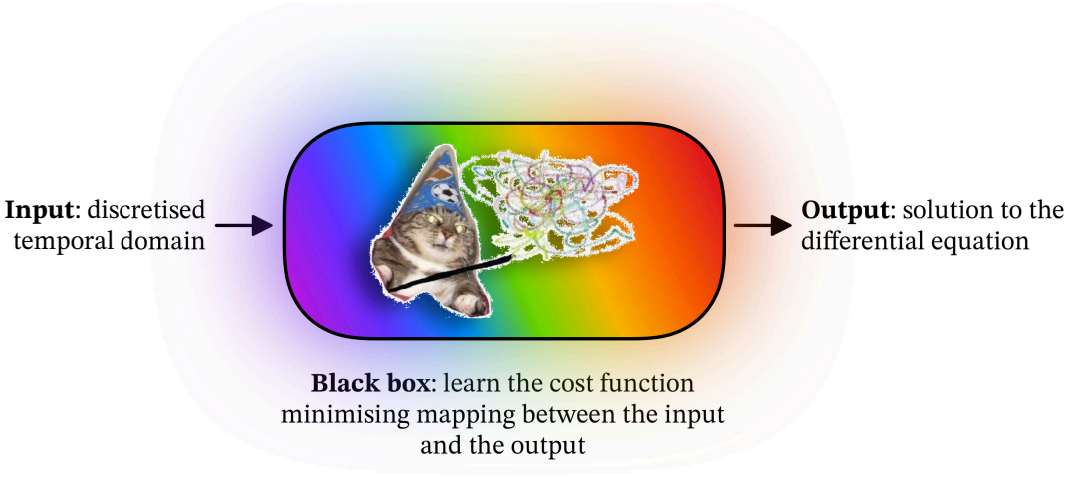


Figure 1.2: A machine learning model to learn the solution of an initial value problem. [“Вжух” cat taken from [Mem17].]

1.3 Machine learning in quantum fluids

Machine learning has been applied in the field of quantum fluids for two overarching purposes: *reinforcement learning*[2] has been used to learn to control experimental procedures [Wig+16; Tra+18; Nak+19; Bar+20; Dav+20; Ven+22; Mil+23], and convolutional neural networks (introduced in § 2.7) for classification-type prob-

[2]Reinforcement learning is like teaching a dog a new trick—the machine learning model acts as an ‘agent’ (the dog), and learns how to interact with its environment (e.g., how to sit) in order to maximise some reward (e.g., to earn a treat). Outside of quantum fluids, reinforcement learning has been used to train computers to play games [Mni+13; Sil+16] and to train robots to complete real-world tasks [Ope+19; Ha+20], for example. We do not use reinforcement learning techniques in this thesis; for further reading, see *Reinforcement Learning: An Introduction* by Sutton and Barto [SB18].

lems [Nes+20; Guo+21; Met+21; Kee+23]. Machine learning techniques enable researchers to automate manual tasks and uncover new physics more efficiently. Let us briefly outline the key results of these classification-type problems.

Ness, *et al.* introduce a novel method to simplify the pipeline of absorption imaging in experiments with ultracold atoms. The method reduces noise levels in the captured images by inputting a single exposure to the machine learning model to generate a reference frame, leading to a more accurate extraction of physical observables.

Guo, *et al.* introduce a machine learning model which can identify the presence and, if appropriate, the location of dark solitons. Human-labelled atomic density profiles produced in an experiment are passed through a convolutional neural network during the training procedure. Testing the trained model to determine whether a soliton was present or not gave an accuracy of around 87%. If solitons are predicted, a least-squares fit is performed to determine their location (so this is not predicted by the model, but is instead performed in the post-processing of the test data).

Metz, *et al.* describe a machine learning model used to identify the locations of vortices in atomic density samples. Using the density or density and phase information, one can accurately detect the locations of vortices (or anti-vortices if phase information is available). This machine learning model is already being used to automate an otherwise laborious, manual task in work by Kim, *et al.*, [Kim+22] and Rabga, *et al.*, [Rab+23].

Keepfer, *et al.* introduce a machine learning approach for the 3D reconstruction of superfluid vortex filaments, which is critical in understanding quantum turbulence. This work achieves accurate topological reconstructions from two-dimensional integrated density profiles using convolutional neural networks trained on simulated atomic density images of a Bose–Einstein condensate. This study may enable precise analysis and validation of theoretical predictions in the turbulent dynamics of superfluid vortices.

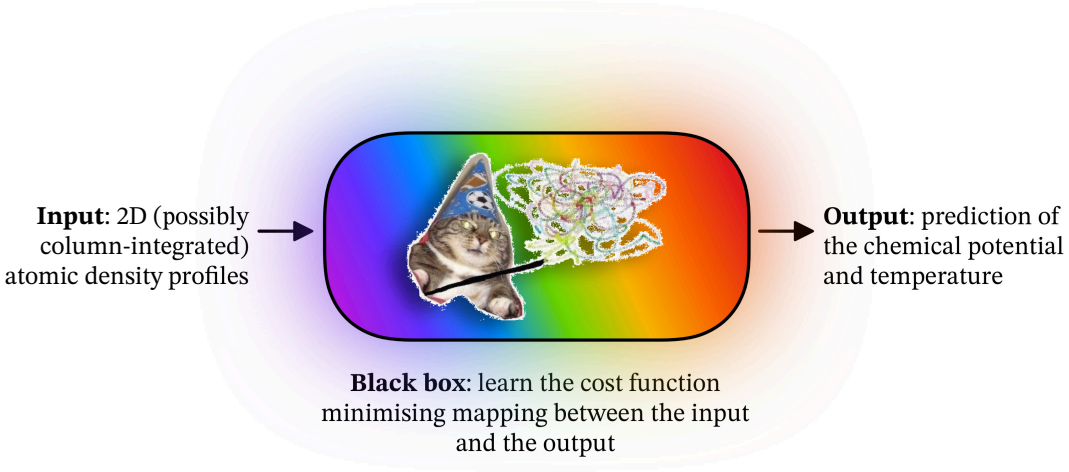


Figure 1.3: A machine learning model to predict the chemical potential and temperature of a Bose gas with a thermal component. ["Вжух" cat taken from [Mem17].]

A current bottleneck in our field's interplay of computational and experimental physics is the identification of the chemical potential and temperature of a Bose gas, necessary for all numerical schemes describing finite temperature gases. Experimentalists rarely need accurate knowledge of these parameters, and they are cumbersome to determine: there is no single formula for the chemical potential (it must be calculated for all geometries of the gas) and measuring the temperature requires experimentalists to destroy their gas during a time-of-flight expansion. We introduce a convolutional neural network which can accurately predict the chemical potential and temperature of thermalised or near-thermalised Bose gases in harmonic and toroidal geometries, given only a two-dimensional atomic density profile (which may be column-integrated). Our model predicts these parameters within fractions of a second, and could be used to give real-time predictions of these parameters in experimental setups. Figure 1.3 shows a black box neural network for our machine learning model.

Part II

Introductions

2

MACHINE LEARNING

PHYSICISTS are well-trained to understand the mathematics of deep learning algorithms, for they are nothing more than an interplay of linear algebra, multivariate calculus, and elementary statistics. For most machine learning practitioners, only knowledge of basic programming (e.g., in Python, Julia or Swift) is required—all of the algorithms to develop a machine learning model are available as high-level abstractions in libraries such as PyTorch [Pas+19], Tensorflow [Mar+15] (Python libraries), Flux [Inn18] (a Julia library) and Core ML [App24] (a Swift library).

This thesis is guided by the application of machine learning to solve initial value, boundary value, and Sturm–Liouville problems in classical and quantum physics, and to predict thermodynamical parameters of Bose gases. In the early stages of our research into solving initial value problems with machine learning, we discovered many deviations between theoretically sound hypotheses and the results obtained through our modelling. This motivated the development of a consistent nomenclature to aid our understanding of machine learning algorithms, particularly as they apply to solving initial value problems in their strictest sense. In this chapter, we elucidate how machines learn and uncover the black box using our nomenclature.

There are two presentations of neural networks in this thesis, both starting from modelling linear relationships. In this chapter, we present a treatment of a simple neural network from its initialisation to a trained model ready for deployment. We additionally introduce the following new contributions to the field: probabilistic activation functions in § 2.2.5, the coupled neural network in § 2.6, and new activation

functions for complex-valued neural networks in § 2.8. We derive these results from first principles by starting with linear regression, a staple of the physicist's toolkit. In part III, chapter 4, we motivate the need for machine learning and deep learning through several examples from classical mechanics.

2.1 Linear regression as a neural network

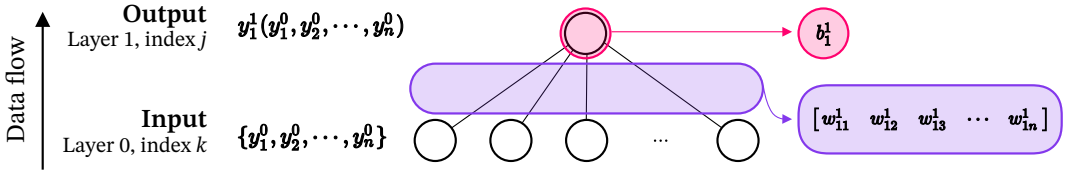


Figure 2.1: A network graph (the 'neural network') illustrates a linear regression model, serving as a geometric interpretation of a hyperplane in feature space. The circles, or neurons, represent either the inputs $y_k^0 \in \mathbb{R}$ or the output $y_1^1 \in \mathbb{R}$. Data flows from bottom to top.

Linear regression can be described using notation and terminology from graph theory. Figure 2.1 shows a graph with two layers: an input layer (layer $\ell = 0$) represents the n independent variables $y_k^{\ell=0}$ (in machine learning, the independent variables are also known as features) and an output layer (layer $\ell = 1$) which represents the dependent variable $y_{j=1}^{\ell=1}$. Every independent variable is represented by a node in the input layer of the network graph. The dependent variable is also represented by a node in the output layer of the network graph. The linear relationship between the dependent and independent variables are visualised as a hyperplane in the n-dimensional feature space.[1] Some features are more important than others, so we weight each feature. We denote each weight by w_{jk}^{ℓ}, which is the weight from a neuron k in layer $\ell - 1$ to a neuron j in layer ℓ; this notation adds a directionality to the graph, where data flows from layer $\ell - 1$ to layer ℓ. Figure 2.3 demonstrates this notation for a deep neural network, which we will introduce in § 2.2. In order to translate the hyperplane up and down, we apply a bias, $b_{j=1}^{\ell=1}$, to the neuron in

[1] A hyperplane is essentially an $n-1$ dimensional subspace, generalising the concept of a line in 2D and a plane in 3D to an n-dimensional setting.

the output layer. In machine learning, this graph is referred to as a neural network. A neural network is a weighted, directed graph composed of nodes connected by edges, where each node is referred to as a neuron. The neuron in the output layer computes the dependent variable

$$y_1^1 = w_{11}^1 y_1^0 + w_{12}^1 y_2^0 + \cdots + w_{1n}^1 y_n^0 + b_1^1 = \sum_{k=1}^{n} w_{1k}^1 y_k^0 + b_1^1, \tag{2.1}$$

where $\{w_{1k}^1\}$ constitutes a set of n weights that also serve as the normal vector to the hyperplane, and b_1^1 is a bias (or offset) that translates the hyperplane along this normal vector. In a neural network, these weights and (possibly many) biases, collectively denoted as $\boldsymbol{\theta}$, are referred to as the network parameters.

To quantify the efficacy of the prediction in equation (2.1) we compute the half-square error for each observation, i, passed through our regression. Since this metric is valid not just for the linear regression, we will drop all indices and denote $\hat{y}_i$ as the predicted value of the ith observation and y_i as its expected value. The half-square error is

$$\mathcal{c}_i(\boldsymbol{\theta}) = \frac{1}{2}(\hat{y}_i - y_i)^2. \tag{2.2}$$

Assuming N total observations, we generalise this to an average measure over all observations using the mean square error,

$$\mathcal{C}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{c}_i(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{2}(\hat{y}_i - y_i)^2. \tag{2.3}$$

The mean square error is an example of a cost function—a comparative metric between the expected solution and the predicted solution by the neural network.[2] The optimisation task is then to find the hyperplane, defined by parameters $\boldsymbol{\theta}$, that minimises the cost function $\mathcal{C}$.

[2] In the literature, the cost function may also be referred to as a loss or objective function. The loss function may also use the notation $\mathcal{L}$, which we have decided not to use to avoid confusion with a discussion of Hamilton's principle in part III, chapter 4. We will always use the notation $\mathcal{C}$ and we will always talk about a cost function.

2.2 Building a deep neural network

2.2.1 Nomenclature

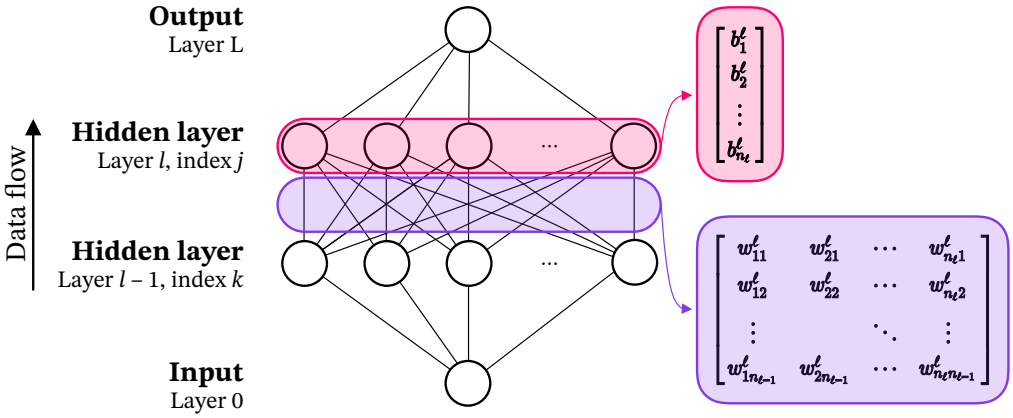


Figure 2.2: A fully connected and feedforward deep neural network is shown. Information flows from bottom to top. Every connection between every neuron is associated with a weight. Every neuron (except the input layer) has an associated bias, shifting its value up and down. The weights between every layer are a matrix, and the biases for every neuron in a layer are a vector. The input and output layers in this figure show only one neuron, but in general there could be many, representing for example pixels of an image or data points.

Real-world data is often non-linear, such as those in image recognition and classification, natural language processing, and in many physical phenomena. There is a need to increase the representational capacity of the neural network to describe non-linear phenomena, which a linear regression would obviously be unable to model. This is achieved by adding intermediate layers between the input and the output, and by applying a non-linear function, $\mathcal{A}(x)$, to each non-input neuron.[3] In machine learning, the non-linear functions are called *activation functions*. With this improved representational capacity, our models can now learn to identify and represent complex patterns and relationships within the data. This is *deep learning*; Figure 2.2 shows a deep neural network.

[3]Taking successive linear layers is equivalent to just one linear operation—everything collapses into a linear model such that no non-linear problem can be learnt.

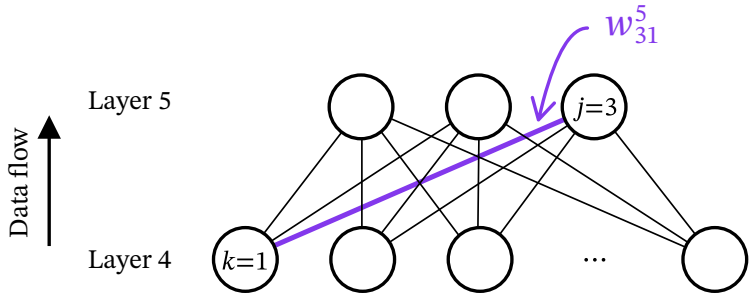


Figure 2.3: Two layers of a fully connected and feedforward neural network. Data flows from bottom to top. A connection between the first neuron of the fourth layer and the third neuron of the fifth layer is highlighted (purple, thick line). This connection has a weight $w_{jk}^{\ell} = w_{31}^5$. In layer ℓ, weights are directed to j from k.

Assume that the deep neural network has L layers (where the input layer is $\ell = 0$). For an arbitrary layer ℓ in the network, let n_ℓ be the number of neurons in that layer, and $n_{\ell-1}$ be the number of neurons in the $(\ell-1)$th layer. The number of neurons in a given layer is referred to as the width of that layer. The number of layers in a neural network is the depth of the network (hence, *deep* learning).

A fully connected neural network is a network where all $n_{\ell-1}$ neurons in layer $\ell-1$ are input into all n_ℓ neurons in layer ℓ (this is analogous to a sequence of complete bipartite graphs). A feedforward neural network is a network where data moves only in one direction without any loops or recurrent connections (this is analogous to a directed acyclic graph). Unless otherwise stated, all networks that we consider in this thesis are fully connected and feedforward.

Let the kth neuron from the $(\ell-1)$th layer have an output value of $y_k^{\ell-1}$ and let the jth neuron from the ℓth layer have an output value of y_j^{ℓ}. Let w_{jk}^{ℓ} be the weight from the kth neuron in the $(\ell-1)$th layer to the jth neuron in the ℓth layer. The order of the k and j indices may not appear to be a natural choice; such ordering is important for clarity in the discussion of optimisation algorithms in § 2.5. Figure 2.3 shows the notation for a weight between two layers of a neural network.

2.2.2 Output of a neuron

Using this notation, the output, y_j^ℓ, of the jth neuron in the ℓth layer is

$$z_j^\ell = \sum_{k=1}^{n_{\ell-1}} w_{jk}^\ell y_k^{\ell-1} + b_j^\ell, \tag{2.4a}$$

$$y_j^\ell = \mathcal{A}\left(z_j^\ell\right), \tag{2.4b}$$

where equation (2.4a) is akin to a linear regression and equation (2.4b) is referred to as the activation or output of a neuron in the deep neural network.

2.2.3 Hidden layers

The layers in between the input and the output are called hidden layers. The hidden layers are used to learn a mapping between the input and the output. Unlike the input and output layers, which have clear meanings related to the data and the task, the activations of hidden neurons don't necessarily correspond to easily interpretable concepts. Hidden layers often involve high-dimensional spaces and complex non-linear transformations—there is no clear semantic meaning of the hidden layers. The number of hidden layers and the number of neurons in each layer are two examples of hyperparameters—parameters which describe the network architecture. There is no way to know *a priori* what the values of such hyperparameters should be, and these should be fine-tuned during the training of machine learning models.

2.2.4 Activation functions

Activation functions are introduced between layers in order to obey the *universal approximation theorem* [HSW89; Pin99; Les+93]—a statement that a possibly infinitely wide neural network trained for possibly infinite time can represent any continuous function so long as non-linear activation functions are used.[4] Examples of activation functions in the real domain are $\mathcal{A}(x) = \tanh(x)$, the sigmoid, $\mathcal{A}(x) = (1 + \mathrm{e}^{-x})^{-1}$,

[4] The universal approximation theorems do not tell us *how* to find the optimal weights and biases, only that they exist.

and $\mathcal{A}(x) = \mathrm{ReLU}(x) = \max(0, x)$ [Fuk69]. The hyperbolic tangent is just a rescaled sigmoid function, $\tanh(x) = 2\sigma(2x) - 1$.

In general, it is not obvious *a priori* which activation function to choose for machine learning modelling. An activation function suitable for one problem is not necessarily suitable for a different problem, as a consequence of the *No Free Lunch* theorem [WM97].

As a historical note, it was once accepted that activation functions should be bounded (e.g., akin to a sigmoid or hyperbolic tangent function)—we stress that this is no longer a requirement. Bounded functions often run into issues with saturation near their asymptotes. Optimisation algorithms in machine learning rely on taking the gradient of a cost function with respect to the network parameters in order to descend through the cost landscape. These gradients become approximately zero in large regions of the parameter space, and learning becomes difficult (the optimisation algorithms may get stuck far from the global minimum).

2.2.5 Probabilistic activation functions

Model averaging

Neural networks exhibit run-to-run variation—different initialisation of the network parameters and stochastic descent through the (possibly vast) parameter space mean that each neural network will return a different output compared to another instance of a neural network with the same architecture, cost function, inputs and hyperparameters. It may be the case that some runs fail to learn the mapping between the input and output, whilst others learn a more accurate mapping. It has been noted extensively [Bre96; Die00; HS90; KV94; OM99] that performing an average (either an arithmetic or geometric mean) of many neural networks produces a more accurate model than any individual neural network.

Model averaging may be heuristically achieved using an algorithm called dropout, which we will describe now.

Dropout

Consider the output of the jth neuron in the ℓth layer, given in equations (2.4a) and (2.4b). Dropout, introduced by Srivastava *et al.* [Sri+14], is an algorithm which removes neurons with probability p and preserves neurons with probability $1 - p$ at the start of each epoch (which is a Bernoulli trial). With dropout, the architecture in equations (2.4a) and (2.4b) become

$$r_k^{\ell-1} \sim \text{Bernoulli}(p), \tag{2.5a}$$

$$\tilde{y}_k^{\ell-1} = r_k^{\ell-1} \cdot y_k^{\ell-1}, \tag{2.5b}$$

$$z_j^{\ell} = \sum_{k=1}^{n_{\ell-1}} w_{jk}^{\ell} \tilde{y}_k^{\ell-1} + b_j^{\ell}, \tag{2.5c}$$

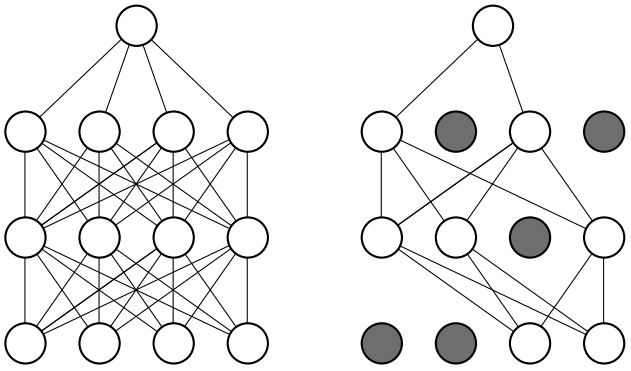
$$y_j^{\ell} = \mathcal{A}(z_j^{\ell}). \tag{2.5d}$$

Figure 2.4 shows a feedforward neural network with and without dropout. These configurations are called subnetworks since they are a subset of the larger network.

Neurons in any layer except the output layer can be involved in the dropout algorithm. For smaller networks, it may be the case that there are no connections between the input and output layer, but this is vanishingly unlikely to be the case for most neural networks (including those in this thesis).

Dropout prevents overfitting (the prevention of overfitting is referred to as regularisation), and also approximately averages the output of many neural networks.

Dropout is performed at the start of each epoch. It is possible to perform a type of dropout at activation time via a class of activation functions which we call *probabilistic activation functions*, extending work by Hendrycks and Gimpel [HG23]. Probabilistic activation functions are in general not a replacement for the dropout algorithm but rather offer comparable results. For learning smooth functions such as solutions to differential equations, probabilistic activation functions are required. For classification problems, dropout is sufficient (and uses less compute time).



(a) (b)

Figure 2.4: (a) A fully connected, feedforward neural network without dropped neurons and (b) the same neural network with dropped neurons (dark grey) and the remaining connections. The neurons which are dropped are stochastically chosen at each epoch.

Dropout and model averaging with probabilistic activation functions

We define the *probabilistic activation function*

$$\mathcal{A}(x) = x\phi(x) = x\int_{-\infty}^{x} \mathrm{d}x'\; p(x'), \tag{2.6}$$

where $p(x)$ is a probability distribution supported on the domain $x \in (-\infty, \infty)$ and the integral defines its cumulative distribution function $\phi(x)$.[5]

Although not smooth or differentiable, it is insightful to consider the Dirac delta distribution, $p(x) = \delta(x)$. The associated cumulative distribution function is the Heaviside step function,

$$\phi(x) = \int_{-\infty}^{x} \mathrm{d}x'\; \delta(x') = \Theta(x), \tag{2.7}$$

[5]Probability distributions with support only on the domain $x \in (0, \infty)$ are not conducive to learning, possibly due to the lack of symmetry. It is our empirical observation that some distributions which are not perfectly symmetric about the origin — such as that used in Mish (see Eq. (2.14c)) — are still conducive to learning.

which leads to the standard (and also non-differentiable) ReLU activation function

$$\mathrm{ReLU}(x) = x\,\Theta(x) = \max(0, x). \tag{2.8}$$

The effect is therefore comparable to a form of dropout; if x is viewed as a random variable, then neurons are either preserved or dropped, with probability 0.5.

More generally, the cumulative distribution function $\phi(x)$ for a continuous, differentiable distribution function $p(x)$ will interpolate smoothly between being close to 0 over much of the initial part of its range, and close to 1 over the latter part of its range. Note also that if $\phi(x)$ is the probability for the outcome of a Bernoulli trial to be $= 1$, it is also the expectation value for the Bernoulli trial. Hence, a probabilistic activation function $\mathcal{A}(x) = x\,\phi(x)$ scales all neuron outputs by the expectation value of a Bernoulli trial, effectively providing something resembling a smooth implementation of dropout and a form of model averaging.

In this way one can derive the Gaussian error linear unit (GELU) activation function [HG23]

$$p(x) = \frac{1}{\rho\sqrt{2\pi}}\exp\left(-\frac{1}{2}\frac{x^2}{\rho^2}\right), \tag{2.9a}$$

$$\phi(x) = \int_{-\infty}^{x} \mathrm{d}x'\,\frac{1}{\rho\sqrt{2\pi}}\exp\left(-\frac{1}{2}\frac{x'^2}{\rho^2}\right) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\rho\sqrt{2}}\right)\right] \tag{2.9b}$$

$$\mathcal{A}(x) = \mathrm{GELU}(x) = x \cdot \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\rho\sqrt{2}}\right)\right]. \tag{2.9c}$$

An activation function we have found in this work to be very useful we call SechLU:

$$p(x) = \frac{1}{2\rho}\mathrm{sech}^2\left(\frac{x}{\rho}\right) \tag{2.10a}$$

$$\phi(x) = \int_{-\infty}^{x} \mathrm{d}x'\,\frac{1}{2\rho}\mathrm{sech}^2\left(\frac{x'}{\rho}\right) = \frac{1}{1 + \exp(-2x/\rho)} \tag{2.10b}$$

$$\mathcal{A}(x) = \mathrm{SechLU}(x) = \frac{x}{1 + \exp(-2x/\rho)}. \tag{2.10c}$$

Since ρ is an adjustable parameter, we can look at the limiting behaviour of SechLU as $\rho \to 0$. The limit approaches 0 from $\pm\infty$—it is double-sided. The two limits are

$$\lim_{\rho\to 0^-} \mathrm{SechLU}(x) = \lim_{\rho\to 0^-} \frac{x}{1+\exp(-2x/\rho)} = 0 \forall x < 0, \tag{2.11a}$$

$$\lim_{\rho\to 0^+} \mathrm{SechLU}(x) = \lim_{\rho\to 0^+} \frac{x}{1+\exp(-2x/\rho)} = x \forall x > 0, \tag{2.11b}$$

which is exactly equivalent to the rectifying linear unit activation function $\mathcal{A}(x) = \mathrm{ReLU}(x) = \max(0, x)$.

Starting with a Cauchy (Lorentzian) distribution, we can similarly obtain an activation function we call CauchyLU:

$$p(x) = \frac{1}{\pi\rho}\left(1 + \frac{x^2}{\rho^2}\right)^{-1}, \tag{2.12a}$$

$$\begin{aligned}\phi(x) &= \int_{-\infty}^{x} \mathrm{d}x' \, \frac{1}{\pi\rho}\left(1 + \frac{x'^2}{\rho^2}\right)^{-1} \\ &= \frac{1}{\pi}\arctan\left(\frac{x}{\rho}\right) + \frac{1}{2}\end{aligned} \tag{2.12b}$$

$$\mathcal{A}(x) = \mathrm{CauchyLU}(x) = \frac{x}{2}\left[1 + \frac{2}{\pi}\arctan\left(\frac{x}{\rho}\right)\right]. \tag{2.12c}$$

Noting it is not everywhere differentiable, we can obtain an activation function which we call LaplaceLU:

$$p(x) = \frac{1}{2\rho}\exp\left(-\frac{|x|}{\rho}\right) \tag{2.13a}$$

$$\begin{aligned}\phi(x) &= \int_{-\infty}^{x} \mathrm{d}x' \, \frac{1}{2\rho}\exp\left(-\frac{|x'|}{\rho}\right) = \begin{cases} \frac{1}{2}\exp\left(\frac{x}{\rho}\right) & \text{if } x < 0 \\ 1 - \frac{1}{2}\exp\left(-\frac{x}{\rho}\right) & \text{if } x \geq 0 \end{cases} \\ &= \frac{1}{2} + \frac{1}{2}\,\mathrm{sgn}(x)\left[1 - \exp\left(-\frac{|x|}{\rho}\right)\right]\end{aligned} \tag{2.13b}$$

$$\mathcal{A}(x) = \mathrm{LaplaceLU}(x) = \frac{x}{2}\left\{1 + \mathrm{sgn}(x)\left[1 - \exp\left(-\frac{|x|}{\rho}\right)\right]\right\}. \tag{2.13c}$$

Finally, the Mish activation function, introduced by Misra in 2019 [Mis20], can also be derived from a (somewhat unusual) probability distribution.

$$p(x) = \frac{1}{\rho}\operatorname{sech}^2\big(\ln\big|1+\mathrm{e}^{x/\rho}\big|\big)\,\frac{\mathrm{e}^{x/\rho}}{1+\mathrm{e}^{x/\rho}}, \tag{2.14a}$$

$$\phi(x) = \int_{-\infty}^{x} \mathrm{d}x'\,\frac{1}{\rho}\operatorname{sech}^2\Big[\ln\big|1+\mathrm{e}^{x'/\rho}\big|\Big]\frac{\mathrm{e}^{x'/\rho}}{1+\mathrm{e}^{x'/\rho}} = \tanh\Big[\ln\big|1+\mathrm{e}^{x/\rho}\big|\Big], \tag{2.14b}$$

$$\mathcal{A}(x) = \operatorname{Mish}(x) = x\tanh\big(\ln\big|1+\mathrm{e}^{x/\rho}\big|\big)\,. \tag{2.14c}$$

The probability distributions, cumulative distributions, activation functions and their derivatives are shown in Figure 2.5.

In all cases, the width parameter ρ may be learnt as an additional hyperparameter by following the same initialisation and optimisation routines as for the weights and biases. We observe faster and more reliable convergence to appropriate physical solutions if we treat ρ as a free parameter.

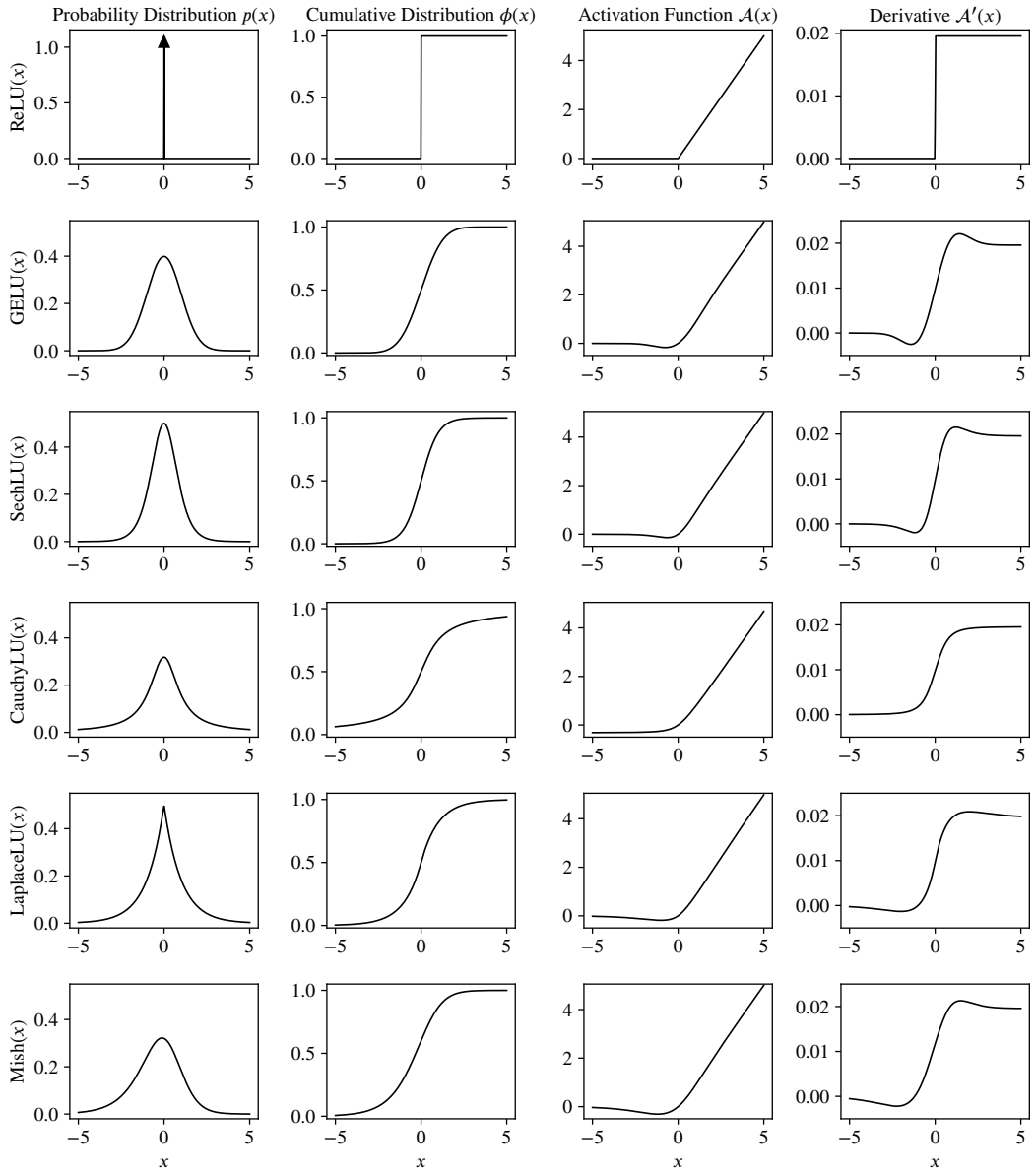


Figure 2.5: A selection of probabilistic activation functions: the ReLU (equation (2.8)), GELU (equation (2.9c)), SechLU (equation (2.10c)), CauchyLU (equation (2.12c)), LaplaceLU (equation (2.13c)) and Mish (equation (2.14c)). All figures use either the variance or scaling parameter $\rho = 1$ (even if this is not the most optimal value). The first column is the probability distribution $p(x)$. The second column is the cumulative distribution $\phi(x) = \int_{-\infty}^{x} \mathrm{d}x' \, p(x')$. The third column is the probabilistic activation function $\mathcal{A}(x) = x\phi(x)$. The fourth column is the derivative of the activation function $\mathcal{A}'(x)$. The functions, with the exception of ReLU, are plotted on common axes to see clearly the relative magnitude and curvature of the functions.

2.3 Initialising a neural network

Before any training or learning takes place, the weights and the biases in the neural network are initialised. This is not a 'best guess' at the solution. The parameter initialisation should not be too large (to avoid diverging values of the gradients) nor too small (to avoid slow training). A standard initialisation algorithm is *Xavier* initialisation [GB10], which defines a distribution from which the weights and biases are drawn from.

Initialisation with constant parameters

This is what not to do.

Consider an arbitrary feedforward, fully connected neural network. Let us assume that the network parameters are all initialised to some constant parameter, ε. If $\varepsilon = 0$, and we have an activation function where $\mathcal{A}_0 = 0$ (e.g., ReLU or tanh), then

$$y_j^\ell = \mathcal{A}(z_j^\ell) = \mathcal{A}\left(\sum_{k=1}^{n_{\ell-1}} w_{jk}^\ell y_k^{\ell-1} + b_j^\ell\right) = \mathcal{A}\left(\sum_{k=1}^{n_{\ell-1}} 0 \cdot y_k^{\ell-1} + 0\right) = \mathcal{A}_0 = 0, \tag{2.15}$$

i.e., all neuron outputs will always be zero.

In § 2.5, we will introduce the optimisation procedures in a neural network—the central idea is to determine the quantities

$$\frac{\partial \mathcal{C}}{\partial w_{jk}^\ell} \quad \text{and} \quad \frac{\partial \mathcal{C}}{\partial b_j^\ell}, \tag{2.16}$$

which is, broadly speaking, the magnitude of how much we should descend through the parameter space to reach a global minimum. The calculation of equation (2.16) is by the chain rule. For the weights,

$$\frac{\partial \mathcal{C}}{\partial w_{jk}^\ell} = \frac{\partial \mathcal{C}}{\partial y_j^\ell} \frac{\partial y_j^\ell}{\partial z_j^\ell} \frac{\partial z_j^\ell}{\partial w_{jk}^\ell}, \tag{2.17}$$

where the biases follow similarly. Without making comments about the other terms in the chain rule, we can immediately note from equation (2.15) that under the assumption that the weights and biases are initialised to the same value,

$$\frac{\partial z_j^\ell}{\partial w_{jk}^\ell} = 0, \tag{2.18}$$

since all neuron outputs are zero—this informs the optimisation algorithm that no step should be taken through the parameter space! If we had an activation function where $\mathcal{A}_0 \neq 0$ (e.g., sigmoid or GELU), then the weight updates would all have the same non-zero magnitude, so every parameter carries the same importance in the network (which is not true in general—some parameters are more important than others). In either case, no learning can take place. Indeed, by corollary, if ε was any constant parameter, then no learning would take place (either because the required change in the weight or bias would be the same or zero for every parameter in the network).

Neural networks do not learn from highly symmetric initialisation schemes. We must break the symmetry in the initialisation procedure, and we can do this by sampling from an underlying probability distribution such as a uniform (rectangular) distribution, $U(-\varepsilon, \varepsilon)$, or a normal distribution, $\mathcal{N}(0, \varepsilon^2)$. The values of ε must be chosen carefully depending on the number of neurons in a layer and the activation function used in that layer of the network.

Generalised Xavier Initialisation

In their original derivation, Glorot and Bengio [GB10] derive an initialisation scheme assuming that hyperbolic tangent functions are used throughout the network. In this section, we derive a generalised approach for any differentiable activation function. Let us ignore any bias terms to simplify the mathematics without loss of generality; the biases are, in any case, drawn from the same distribution as the weights. Assume that the weights distribution has zero mean and variance σ^2. Assume further that

the inputs $y_k^{\ell-1}$ are distributed with zero mean and variance ρ^2 (where, in general, $\sigma^2 \neq \rho^2$).

If the parameters are not initialised with zero mean, then there will be undesirable and erratic zig-zagging through the parameter space during optimisation—it may cause overshooting of the minima. If the parameters are not initialised with constant variance, i.e., $\mathrm{Var}(y_k^{\ell-1}) \approx \mathrm{Var}(y_j^{\ell})$, then the values of the outputs of a layer will decrease exponentially (vanish) or increase exponentially (explode) as you move from bottom to top through the network. The presence of either exploding or vanishing gradients means that the network will be difficult or impossible to train.

Early on in the training, values of z_j^{ℓ} are small [GB10]. Assuming differentiability, many activation functions (such as sigmoid, tanh, and the probabilistic activation functions introduced in § 2.2.5) behave linearly to first order by their Maclaurin expansion,

$$y_j^{\ell} = \mathcal{A}(z_j^{\ell}) \approx \mathcal{A}(0) + \mathcal{A}'(0) z_j^{\ell}. \tag{2.19}$$

where $\mathcal{A}(0) \equiv \mathcal{A}_0$ and $\mathcal{A}'(0) \equiv \mathcal{A}_0'$.

The mean of the inputs is, under the approximation in equation (2.19),

$$\begin{aligned} \mathbb{E}(y_j^{\ell}) &= \mathbb{E}\left[\mathcal{A}\left(\sum_{k=1}^{n_{\ell-1}} w_{jk}^{\ell} y_k^{\ell-1}\right)\right] \\ &\approx \mathbb{E}\left[\mathcal{A}_0 + \mathcal{A}_0' \sum_{k=1}^{n_{\ell-1}} w_{jk}^{\ell} y_k^{\ell-1}\right]. \end{aligned} \tag{2.20}$$

Since the expectation operator is linear, it can be moved into the summation

$$\mathbb{E}(y_j^{\ell}) \approx \mathcal{A}_0 + \mathcal{A}_0' \sum_{k=1}^{n_{\ell-1}} \mathbb{E}\left[w_{jk}^{\ell} y_k^{\ell-1}\right]. \tag{2.21}$$

Assuming that w_{jk}^{ℓ} and $y_k^{\ell-1}$ are independent random variables, one can note that the expected value of a product of independent random variables is the product of the expectation values of those variables

$$\mathbb{E}(y_j^{\ell}) \approx \mathcal{A}_0 + \mathcal{A}_0' \sum_{k=1}^{n_{\ell-1}} \mathbb{E}\big[w_{jk}^{\ell}\big]\mathbb{E}\big[y_k^{\ell-1}\big]. \tag{2.22}$$

Since we originally assumed that the distribution from which the weights are drawn and the distribution of the neuron inputs have zero mean, we immediately note that, $\forall y_j^{\ell}$,

$$\mathbb{E}(y_j^{\ell}) \approx \mathcal{A}_0. \tag{2.23}$$

The variance is

$$\begin{aligned}
\mathrm{Var}[y_j^{\ell}] &= \mathbb{E}[(y_j^{\ell})^2] - \Big(\mathbb{E}[y_j^{\ell}]\Big)^2 && (2.24)\\
&\approx \mathbb{E}\left[\left(\mathcal{A}_0 + \mathcal{A}_0' \sum_{k=1}^{n_{\ell-1}} w_{jk}^{\ell} y_k^{\ell-1}\right)^2\right] - \mathcal{A}_0^2 && (2.25)\\
&= \mathbb{E}\left[\mathcal{A}_0^2 + 2\mathcal{A}_0\mathcal{A}_0' \sum_{k=1}^{n_{\ell-1}} w_{jk}^{\ell} y_k^{\ell-1} + \left(\mathcal{A}_0' \sum_{k=1}^{n_{\ell-1}} w_{jk}^{\ell} y_k^{\ell-1}\right)^2\right] - \mathcal{A}_0^2 && (2.26)\\
&= \mathbb{E}\left[\mathcal{A}_0^2 + 2\mathcal{A}_0\mathcal{A}_0' \sum_{k=1}^{n_{\ell-1}} w_{jk}^{\ell} y_k^{\ell-1} + (\mathcal{A}_0')^2\left(\sum_{k=1}^{n_{\ell-1}} w_{jk}^{\ell} y_k^{\ell-1}\right)^2\right] - \mathcal{A}_0^2 && (2.27)\\
&= \mathcal{A}_0^2 + 2\mathcal{A}_0\mathcal{A}_0' \sum_{k=1}^{n_{\ell-1}} \mathbb{E}[w_{jk}^{\ell} y_k^{\ell-1}] + \mathbb{E}\left[(\mathcal{A}_0')^2\left(\sum_{k=1}^{n_{\ell-1}} w_{jk}^{\ell} y_k^{\ell-1}\right)^2\right] - \mathcal{A}_0^2, && (2.28)
\end{aligned}$$

where in the last line we can again use the assumption that the mean of the neuron outputs is zero and that the weights and the outputs are independent. Since

$(\sum_i a_i)^2 = \sum_i a_i^2 + 2\sum_{i<j} a_i a_j$, care should be taken with the square of the summation. It follows that

$$\mathrm{Var}[y_j^\ell] = \mathcal{A}_0^2 + (\mathcal{A}_0')^2 \mathbb{E}\left[\sum_{k=1}^{n_{\ell-1}} (w_{jk}^\ell y_k^{\ell-1})^2\right] \tag{2.29}$$

$$+ 2(\mathcal{A}_0')^2 \sum_{\substack{k=1 \\ k\neq k'}}^{n_{\ell-1}} \mathbb{E}[w_{jk}^\ell y_k^{\ell-1} w_{jk'}^\ell y_{k'}^{\ell-1}] - \mathcal{A}_0^2 \tag{2.30}$$

$$= \mathcal{A}_0^2 + (\mathcal{A}_0')^2 \sum_{k=1}^{n_{\ell-1}} \mathbb{E}[(w_{jk}^\ell)^2]\mathbb{E}[(y_k^{\ell-1})^2] \tag{2.31}$$

$$+ 2(\mathcal{A}_0')^2 \sum_{\substack{k=1 \\ k\neq k'}}^{n_{\ell-1}} \mathbb{E}[w_{jk}^\ell y_k^{\ell-1} w_{jk'}^\ell y_{k'}^{\ell-1}] - \mathcal{A}_0^2. \tag{2.32}$$

Due to the aforementioned independence and zero mean assumptions, the cross-summation term is zero. The variance therefore simplifies further

$$\mathrm{Var}[y_j^\ell] = \mathcal{A}_0^2 + (\mathcal{A}_0')^2 n_{\ell-1}\sigma^2\rho^2 - \mathcal{A}_0^2 \tag{2.33}$$

$$= (\mathcal{A}_0')^2 n_{\ell-1}\sigma^2\rho^2. \tag{2.34}$$

Note that the variance must be the same from layer to layer, i.e., $(\mathcal{A}_0')^2 n_{\ell-1}\sigma^2 = 1$ and, additionally, $(\mathcal{A}_0')^2 n_\ell \sigma^2 = 1$ (noting that, in general, $n_{\ell-1} \neq n_\ell$). Both of these conditions are satisfied by considering the average of $n_{\ell-1}$ and n_ℓ,

$$\frac{1}{2}\left[(\mathcal{A}_0')^2 n_{\ell-1} + (\mathcal{A}_0')^2 n_\ell\right]\sigma^2 = 1 \implies \sigma^2 = \frac{2}{(\mathcal{A}_0')^2(n_{\ell-1}+n_\ell)}. \tag{2.35}$$

Generalised Xavier initialisation uses a uniform (rectangular) distribution over the interval $[-a, a]$. We wish to find a formula for the uniform distribution in terms of the number of neurons in each layer. In order to find the variance of this distribution, one can use the moment generating functions of the uniform distribution to find the square of the mean values

$$[\mathbb{E}(x)]^2 = \left[\frac{1}{2a}\int_{-a}^{a} x\,\mathrm{d}x\right]^2 = 0, \tag{2.36}$$

and the mean of the squared values

$$\mathbb{E}(x^2) = \frac{1}{2a}\int_{-a}^{a} x^2 \,\mathrm{d}x = \frac{a^3}{3}. \tag{2.37}$$

The variance of the uniform distribution is

$$\sigma^2 = \mathbb{E}(x^2) - [\mathbb{E}(x)]^2 = \frac{a^3}{3}. \tag{2.38}$$

Equating equation (2.35) with equation (2.38) gives the values of the support of the distribution in terms of the number of neurons in each layer

$$w_{jk}^{\ell} \sim U\left(-\sqrt{\frac{6}{(\mathcal{A}_0')^2(n_{\ell-1}+n_\ell)}}, +\sqrt{\frac{6}{(\mathcal{A}_0')^2(n_{\ell-1}+n_\ell)}}\right). \tag{2.39}$$

For $\tanh(x)$, the scaling factor $\mathcal{A}_0' = 1$. For any probabilistic activation function, the scaling factor $\mathcal{A}_0' = \phi(0)$. If the underlying probability distribution is symmetric, then $\mathcal{A}_0' = 1/2$.

Note that the initialisation we employ for linear layers (where equation (2.39) would be undefined since $\mathcal{A}_0' = 0$) is the Kaiming scheme, where one samples from an unscaled uniform distribution [He+15]:

$$w_{jk}^{\ell} \sim U\left(-\sqrt{\frac{1}{n_\ell}}, \sqrt{\frac{1}{n_\ell}}\right). \tag{2.40}$$

The benefits of choosing either a Gaussian (normal) distribution or a uniform distribution have yet to be studied extensively. The scaling and range of the distributions are, however, important for the outcome of the optimisation procedure and for the model's ability to generalise [GBC16].

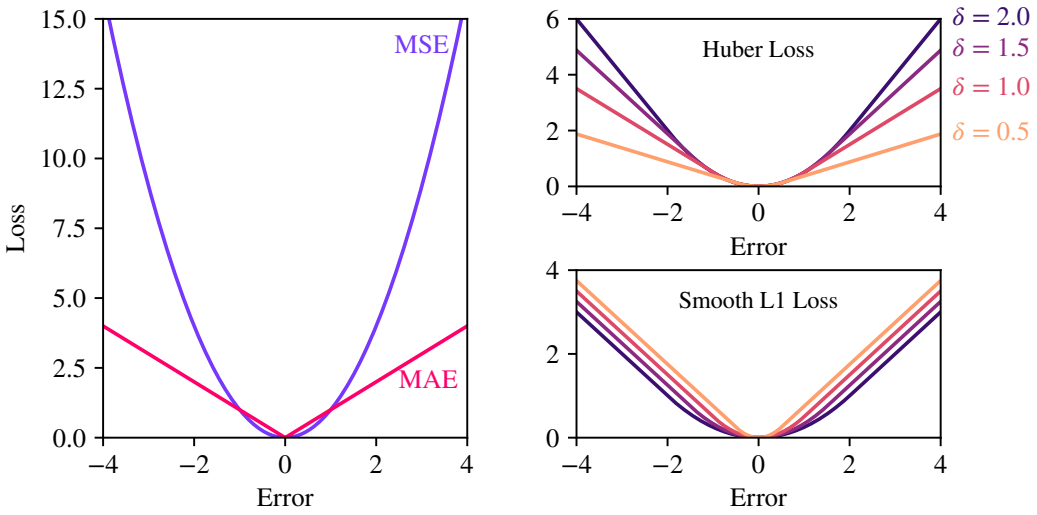


Figure 2.6: **Left**: the mean square error (MSE) and mean absolute error (MAE) for various values of the error. **Right**: the Huber loss (top) and the Smooth L1 Loss (bottom) for various values of δ. For the Huber loss, the values of δ are decreasing from top to bottom. For the Smooth L1 Loss, the values of δ are increasing from top to bottom due to the division by δ.

2.4 Cost functions

Cost functions are a metric used to evaluate the performance of a machine learning model against some criteria, such as how well it classifies data that is input to it. We have already met the mean square error when discussing linear regression in § 2.1. The mean square error is smooth and differentiable everywhere, which makes it ideal for the gradient-based nature of the learning algorithms. Its quadratic nature penalises large errors more than smaller ones but can also cause the model to focus too extensively on outliers during training. In order to treat outliers more effectively, one may wish to treat them the same as any other data point. The mean absolute error measures the average of the absolute errors

$$\mathcal{C}_{\text{MAE}}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} |y^i_{\boldsymbol{\theta}} - y^i|. \tag{2.41}$$

The mean absolute error is not differentiable at zero.

A combination of the mean square error and mean absolute error—known as the Huber loss [Hub64]—is sometimes used to treat outliers more effectively like a mean absolute error and penalise large errors like a mean-square error. It also removes the differentiability at zero issues. The Huber loss is defined as

$$\mathcal{C}_{\text{Huber}}(\boldsymbol{\theta}) = \begin{cases} \frac{1}{2}(y^i_{\boldsymbol{\theta}} - y^i)^2 & \text{for } |y^i_{\boldsymbol{\theta}} - y^i| \leq \delta, \\ \delta|y^i_{\boldsymbol{\theta}} - y^i| - \frac{1}{2}\delta^2 & \text{otherwise,} \end{cases} \tag{2.42}$$

where δ is an adjustable hyperparameter.

One can also define the smooth L1 loss[6] as

$$\mathcal{C}_{\text{SmoothL1}}(\boldsymbol{\theta}) = \frac{1}{\delta}\mathcal{C}_{\text{Huber}}(\boldsymbol{\theta}). \tag{2.43}$$

When $\delta \to 0$, $\mathcal{C}_{\text{SmoothL1}}(\boldsymbol{\theta}) \to \mathcal{C}_{\text{MAE}}(\boldsymbol{\theta})$, whereas $\mathcal{C}_{\text{Huber}}(\boldsymbol{\theta}) \to 0$. When $\delta \to \infty$, $\mathcal{C}_{\text{SmoothL1}}(\boldsymbol{\theta}) \to 0$, whereas $\mathcal{C}_{\text{Huber}}(\boldsymbol{\theta}) \to \mathcal{C}_{\text{MSE}}(\boldsymbol{\theta})$.

The mean square error, mean absolute error, and Huber loss are shown in Figure 2.6. The smooth L1 loss has also been demonstrated in some machine learning models to prevent exploding gradients [Gir15].

A cost function may also be a sum of many terms (depending on your prior knowledge of the problem), although we stress through our own experiments and through earlier research of Vapnik [Vap95; Vap82], the principle of parsimony should guide cost function design, avoiding unnecessary complexity that may lead to overfitting or hinder training.

2.5 Learning from a deep neural network

The machine learning problem is to find the set of weights and biases which provide a cost function minimising mapping between whatever data is input and the desired output. The state of the art optimisation algorithm (which we also use throughout

[6]We have not made this unimportant distinction in notation: the 'L1' loss is sometimes referred to as the mean absolute error, and the 'L2' loss is sometimes referred to as the mean square error.

this thesis) is the adaptive moment estimation or *Adam* optimisation algorithm [KB17], which we will derive now. The Adam algorithm is the descendant of other optimisation algorithms that have emerged over the last few decades. We will follow Adam's lineage, starting with gradient descent, before arriving at the equations for its implementation.

Equation (2.4b) states that the output of the neural network is explicitly dependent upon the weights, biases and values of neurons in the previous layer (the weighted, shifted sum of these are passed through a non-linearity such as the probabilistic activations discussed in § 2.2.5).

2.5.1 Gradient descent

Gradient descent—usually attributed to Cauchy—is a prototypical algorithm for adjusting all the weights and biases in the neural network (i.e., to train our neural network). Gradient descent is rarely used in practice due to its slower convergence compared to modern algorithms, but most common learning algorithms are derived from it, so we shall provide a derivation of it here.

All cost functions are parameterised their weights and biases, $\boldsymbol{\theta}$. Let us again assume there are no biases in the network and focus on the adjustments required for the weights. For an L-layer neural network,

$$\mathcal{C}(\boldsymbol{\theta}) \equiv \mathcal{C}(w^1_{11}, \cdots, w^\ell_{n_\ell n_{\ell-1}}, \cdots, w^L_{n_L n_{L-1}}). \tag{2.44}$$

We wish to adjust the weights by some amount Δw^ℓ_{jk} (and, in general, the biases by some amount Δb^ℓ_j), where the adjustment is assumed to be small (to avoid overshooting the global minima). All adjustments are described by the vector $\Delta\boldsymbol{\theta}$. Taylor expanding the cost function to first-order about $\Delta\boldsymbol{\theta}$,

$$\mathcal{C}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \approx \mathcal{C} + (\Delta\boldsymbol{\theta}) \cdot \nabla_{\boldsymbol{\theta}} \mathcal{C}(\boldsymbol{\theta}), \tag{2.45}$$

where

$$\nabla_{\boldsymbol{\theta}}\mathcal{C}(\boldsymbol{\theta}) = \left(\frac{\partial\mathcal{C}(\boldsymbol{\theta})}{\partial w_{11}^{1}}, \cdots, \frac{\partial\mathcal{C}(\boldsymbol{\theta})}{\partial w_{n_\ell n_{\ell-1}}^{\ell}}, \cdots, \frac{\partial\mathcal{C}(\boldsymbol{\theta})}{\partial w_{n_L n_{L-1}}^{L}}\right). \tag{2.46}$$

For $\mathcal{C} \to 0$, we require that, in general, $\mathcal{C}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) < \mathcal{C}(\boldsymbol{\theta})$, and that the adjustments point in the opposite direction to the gradient, i.e.,

$$\Delta\boldsymbol{\theta} = -\eta\nabla_{\boldsymbol{\theta}}\mathcal{C}(\boldsymbol{\theta}), \tag{2.47}$$

where $\eta > 0$ is (so far) a fixed global hyperparameter (a parameter which describes a neural network) known as the step size or learning rate. One, therefore, has the epoch-to-epoch update formula for the weights,

$$\Delta w_{jk}^{\ell} = -\eta\frac{\partial\mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}, \tag{2.48}$$

and, by corollary, for the biases,

$$\Delta b_{j}^{\ell} = -\eta\frac{\partial\mathcal{C}(\boldsymbol{\theta})}{\partial b_{j}^{\ell}}. \tag{2.49}$$

Equation (2.48) and equation (2.49) describe the gradient descent throughout the parameter space. For simple cost functions, the weight and bias update formulae can be analytically determined; the update formulae are otherwise determined by *automatic differentiation*.

Dimensional interpretation of neural network parameters

In a neural network, each parameter—be it a weight w, a bias b, or any other parameter λ—can be interpreted as contributing a new dimension to the cost landscape which we have denoted as $\mathcal{C}(\boldsymbol{\theta})$. The vector $\boldsymbol{\theta}$ is the space in which optimisation occurs (the *parameter space*).

For a network with N parameters, we can envision an $(N+1)$-dimensional space: N dimensions for the parameters, and one for the cost value. Each point in this space represents a unique configuration of the network, with the corresponding cost value.

The gradient $\nabla_{\boldsymbol{\theta}} \mathcal{C}(\boldsymbol{\theta})$ in equation (2.46) is then a vector in this (highly non-convex) N-dimensional parameter space. Gradient descent may then be understood as navigating this high-dimensional landscape, seeking the global minimum by iteratively moving towards lower-cost regions.

Gradient descent with momentum

Gradient descent will take a long time to traverse flat regions of the parameter space where parts of the gradient of the cost function are approximately zero. Whilst gradient descent will deterministically reach the global minimum, convergence is slow. To accelerate the descent through regions of approximately zero gradient, one can add a momentum term[7] to equation (2.48) and equation (2.49), which is dependent upon the weight change in the previous epoch. We define a velocity v_{jk}^{ℓ} in addition to our parameter update between the $(\mathcal{E}-1)$th epoch to the $\mathcal{E}$th epoch,

$$\begin{aligned} v_{jk}^{\ell} &= \beta v_{jk}^{\ell} + \eta \frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}, \\ \Delta w_{jk}^{\ell} &= -v_{jk}^{\ell}, \end{aligned} \tag{2.50}$$

where $\beta \in [0,1]$ is an adjustable momentum hyperparameter [Sut+13; Ben12] and $\Delta w_{jk}^{\ell} = 0$ at epoch 0 (since no changes are made until epoch 1).

Stochastic gradient descent and batching

Let us assume that we have some training dataset which contains N training examples (e.g., one training example may be one atomic density profile of a Bose–Einstein

[7] Akin to giving a ball a push down a hill, momentum in gradient descent acts to give the change in weights and biases a 'push' through the cost landscape.

condensate with two labels: a chemical potential and temperature). The number of training examples may be arbitrarily large.

In our previous discussion of gradient descent in section § 2.5.1, we did not comment about the size of the dataset. Let us extend equations (2.48) and (2.49) to include the size of the dataset. By taking an average over the entire dataset, the change in a given weight is

$$\Delta w_{jk}^{\ell} = -\frac{1}{N}\sum_{i=1}^{N} \eta \frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}, \tag{2.51}$$

and the change in a bias is

$$\Delta b_{j}^{\ell} = -\frac{1}{N}\sum_{i=1}^{N} \eta \frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial b_{j}^{\ell}}. \tag{2.52}$$

This produces a single, smooth, deterministic path towards the minimum of the cost function, provided the initial parameters are the same. It is also a computationally terrible idea for large datasets! Every time we wish to take one step closer to the global minimum, we need to re-evaluate the whole training dataset again. This algorithm is referred to as *batch gradient descent*, and we shall swiftly ignore it.

In reality, we do something slightly different. Let us define one *epoch* as one forward pass and one backward pass (i.e., a complete pass) through all N training examples. Let us also define a batch size β as a subset of the N training examples (typical values range from 16–256). Let us finally define the number of iterations as the number of complete passes through the β batches. Therefore, it takes N/β iterations to complete one epoch. This algorithm is referred to as *mini-batch gradient descent*. The change in a given weight is

$$\Delta w_{jk}^{\ell} = -\frac{1}{\beta}\sum_{i=1}^{\beta} \eta \frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}, \tag{2.53}$$

and the change in a bias is

$$\Delta b_j^\ell = -\frac{1}{\beta}\sum_{i=1}^{\beta}\eta\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial b_j^\ell}, \tag{2.54}$$

It is computationally easier to explore subsets of the overall training data. Convergence in parameter space can occur faster in wall-clock time with smaller batches due to the more frequent updates, although the path to convergence may be noisier compared to using the full batch. Machine learning libraries also use efficient vectorisation techniques to calculate equations (2.53) and (2.54).

If $\beta = N$, we recover batch gradient descent. If $\beta = 1$, we discover *stochastic gradient descent*. A smaller batch size generally improves the generalisation capabilities of the neural network [Kes+17; ML18] but increases the computation required. There are diminishing returns in adding more training examples into each batch, although some studies argue that one can obtain better generalisability of models trained on larger batches [HHS18; Goy+18; Smi+18].

Batching is used in most modern optimisation algorithms, which we will explore in §§ 2.5.2–2.5.3.

Stochastic gradient descent and stochastic dynamics

Stochastic gradient descent can be interpreted as a form of stochastic dynamics (see appendix B). One can interpret the rate of change of the parameters (weights and biases) $\boldsymbol{\theta}$ as a continuous-time gradient flow (i.e., a steepest descent curve)

$$\frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} = -\nabla_{\boldsymbol{\theta}}\mathcal{C}(\boldsymbol{\theta}). \tag{2.55}$$

Approximating the continuous change using an Euler integration scheme (see appendix C.1), one can approximate the next state of the parameters based on the current state of the parameters and the rate of change at the current state, scaled by

a small time step, Δt. In the context of gradient descent, the learning rate, η, acts as the timestep, leading to the update rule

$$\Delta\boldsymbol{\theta} = -\eta \nabla_{\boldsymbol{\theta}} \mathcal{C}(\boldsymbol{\theta}), \tag{2.56}$$

which then leads to the usual weight and bias update equations (2.48) and (2.49).

Equation (2.56) may be interpreted as the deterministic component of a stochastic differential equation presented in, for example, the Langevin formation in equation (B.4). This 'guides' the overall dynamics of the optimisation. The stochastic component, arising from mini-batch optimisation, introduces noise to the updates, similar to the thermal fluctuations in the Langevin formulation; the noise may help to escape local minima.

2.5.2 Adaptive gradient methods (AdaGrad and RMSProp)

Classical momentum use a fixed learning rate for every parameter. However, the cost function may be highly sensitive to specific directions in the parameter space and insensitive to others. A class of learning algorithms (AdaGrad [DHS11], RMSProp [Hin18] and Adam [KB17]) instead introduce adjustable hyperparameters η^{ℓ}_{jk} for every parameter in the neural network to adjust the rate of descent across the parameter space.

AdaGrad adapts the learning rates for each parameter by scaling them by a rolling history of previous gradients r^{ℓ}_{jk},

$$r^{\ell}_{jk} = r^{\ell}_{jk} + \left[\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w^{\ell}_{jk}}\right]^{*}\left[\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w^{\ell}_{jk}}\right], \tag{2.57a}$$

$$\eta^{\ell}_{jk} = \eta \frac{1}{\delta + \sqrt{r^{\ell}_{jk}}}, \tag{2.57b}$$

$$\Delta w^{\ell}_{jk} = -\eta^{\ell}_{jk}\left[\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w^{\ell}_{jk}}\right], \tag{2.57c}$$

where * denotes the complex conjugate in the case of complex-valued neural networks, η is the *global* learning rate, and $\delta \sim 10^{-8}$ is a small parameter for numerical stability. The initial values of r_{jk}^{ℓ} are all zero.

AdaGrad sometimes decreases individual parameter learning rates too quickly since gradients are accumulated from the beginning of the training. RMSProp introduces a decay rate for the accumulated gradients, reducing the dependence on earlier gradients and prioritising the more recent history of the descent through the parameter space (e.g., when RMSProp finds a highly convex part of the parameter space, it will very rapidly converge on the minimum, whereas AdaGrad may never make it into such a region of the parameter space). Let $\rho_1 \in [0, 1)$ be a decay parameter. The RMSProp algorithm is

$$r_{jk}^{\ell} = \rho_1 r_{jk}^{\ell} + (1 - \rho_1)\left[\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}\right]^*\left[\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}\right], \tag{2.58a}$$

$$\eta_{jk}^{\ell} = \eta\frac{1}{\delta + \sqrt{r_{jk}^{\ell}}}, \tag{2.58b}$$

$$\Delta w_{jk}^{\ell} = -\eta_{jk}^{\ell}\left[\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}\right]. \tag{2.58c}$$

2.5.3 Adaptive moment estimation (Adam)

Adaptive moment estimation (Adam) is an extension of RMSProp. In addition to the accumulation of the square of the gradients (or square of the modulus of the

gradients), Adam also computes an accumulation of the gradients, decayed by some parameter ρ_2. Thus,

$$r_{jk}^{\ell} = \rho_1 r_{jk}^{\ell} + (1-\rho_1)\left[\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}\right]^{*}\left[\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}\right], \tag{2.59a}$$

$$s_{jk}^{\ell} = \rho_2 s_{jk}^{\ell} + (1-\rho_2)\left[\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}\right], \tag{2.59b}$$

$$\Delta w_{jk}^{\ell} = -\eta \frac{s_{jk}^{\ell}}{\delta + \sqrt{r_{jk}^{\ell}}}, \tag{2.59c}$$

where r_{jk}^{ℓ} and s_{jk}^{ℓ} are both initialised to be zero at epoch 0. Since the accumulated gradient and square of the gradients are initially zero, there is a tendency for training to be biased towards zero values, which is undesirable. Adam introduces correction terms to r_{jk}^{ℓ} and s_{jk}^{ℓ} to avoid such bias,

$$r_{jk}^{\ell} = \rho_1 r_{jk}^{\ell} + (1-\rho_1)\left[\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}\right]^{*}\left[\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}\right], \tag{2.60a}$$

$$s_{jk}^{\ell} = \rho_2 s_{jk}^{\ell} + (1-\rho_2)\left[\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial w_{jk}^{\ell}}\right], \tag{2.60b}$$

$$\hat{r}_{jk}^{\ell} = \frac{r_{jk}^{\ell}}{1-\rho_1^{\mathcal{E}}} \tag{2.60c}$$

$$\hat{s}_{jk}^{\ell} = \frac{s_{jk}^{\ell}}{1-\rho_2^{\mathcal{E}}} \tag{2.60d}$$

$$\Delta w_{jk}^{\ell} = -\eta \frac{\hat{s}_{jk}^{\ell}}{\delta + \sqrt{\hat{r}_{jk}^{\ell}}}, \tag{2.60e}$$

where equations (2.60c) and (2.60d) are the corrected accumulated gradients and $\rho_i^{\mathcal{E}}$ is the decay parameter raised to power of the current epoch number, $\mathcal{E}$. As with gradient descent, the biases are updated in a similar way. This completes the Adam optimisation algorithm. It is the current state of the art optimisation algorithm in the machine learning literature.

Loshchilov and Hutter [LH19] propose AdamW, which adds a weight decay term to Δw_{jk}^{ℓ}, and they note that this improves the generalisation capabilities of Adam. For our work, we did not notice any significant improvement in learning or generalisation capabilities possible because our datasets are small. We therefore only work with Adam for marginally improved computational time.

2.5.4 Backwards propagation of errors with scalar-valued neurons

In all learning algorithms—from gradient descent to Adam—we need knowledge of the quantities

$$\frac{\partial \mathcal{C}}{\partial w_{jk}^{\ell}} \quad \text{and} \quad \frac{\partial \mathcal{C}}{\partial b_{j}^{\ell}}, \tag{2.61}$$

which we refer to as the *sensitivity* of the cost function with respect to a given weight or bias in the network. When we have completed a forward pass through the network, we have a number of outputs which need evaluating against our already defined cost function. The output layer neuron(s) are dependent upon all neurons which come before it. Since a neural network is a composition of many functions through the network, we note that—from the definitions of the pre-activation in equation (2.4a) and the post-activation in equation (2.4b)—the quantities in equation (2.61) can be determined from the chain rule and simplified,

$$\frac{\partial \mathcal{C}}{\partial w_{jk}^{\ell}} = \frac{\partial \mathcal{C}}{\partial z_{j}^{\ell}} \frac{\partial z_{j}^{\ell}}{\partial w_{jk}^{\ell}} = \frac{\partial \mathcal{C}}{\partial z_{j}^{\ell}} \cdot y_{k}^{\ell-1}, \tag{2.62}$$

and

$$\frac{\partial \mathcal{C}}{\partial b_{j}^{\ell}} = \frac{\partial \mathcal{C}}{\partial z_{j}^{\ell}} \frac{\partial z_{j}^{\ell}}{\partial b_{j}^{\ell}} = \frac{\partial \mathcal{C}}{\partial z_{j}^{\ell}}. \tag{2.63}$$

We now seek an equation for $\partial \mathcal{C} / \partial z_{j}^{\ell}$. This quantity compares a neuron's pre-activation against the cost function and is used to make adjustments to the weights and biases in the network. We will continue with the convention that neurons with index k feed into neurons with index j.

We can express the sensitivity in the output layer, L, by looking at how small or large the rate of change of the cost function is with respect to the weighted input z_j^L (we could, in principle, look at the rate of change with respect to the full output of the neuron, but this involves an extra computational step for the activation function—the weighted input contains all the information we need). Thus,

$$\frac{\partial \mathcal{C}}{\partial z_j^L} = \frac{\partial \mathcal{C}}{\partial y_j^L}\frac{\partial y_j^L}{\partial z_j^L} = \frac{\partial \mathcal{C}}{\partial y_j^L}\mathcal{A}'(z_j^L). \tag{2.64}$$

If equation (2.64) is small, then the neuron is nearly optimal.

We then work top to bottom through the neural network (hence, backpropagation) in order to determine the optimality of each neuron. For the next layer, $L-1$, the sensitivity of the cost function with respect to the neuron pre-activations is

$$\frac{\partial \mathcal{C}}{\partial z_k^{L-1}} = \sum_{j=1}^{n_L} \frac{\partial \mathcal{C}}{\partial z_j^L}\frac{\partial z_j^L}{\partial z_k^{L-1}}, \tag{2.65}$$

where

$$z_j^L = \sum_{k=1}^{n_{L-1}} w_{jk}^L \mathcal{A}(z_k^{L-1}) + b_j^L. \tag{2.66}$$

Using equation (2.66) in equation (2.65) gives the sensitivity of the cost function with respect to a neuron's pre-activation in the penultimate layer

$$\frac{\partial \mathcal{C}}{\partial z_k^{L-1}} = \sum_{j=1}^{n_L} \frac{\partial \mathcal{C}}{\partial z_j^L} w_{jk}^L \mathcal{A}'(z_k^{L-1}). \tag{2.67}$$

Then, in any arbitrary layer $\ell-1$, $\ell = \{1, \cdots, L-2\}$, the sensitivity of the cost function with respect to any arbitrary neuron in that layer is

$$\frac{\partial \mathcal{C}}{\partial z_k^{\ell-1}} = \sum_{j=1}^{n_\ell} \frac{\partial \mathcal{C}}{\partial z_j^{\ell}} w_{jk}^{\ell} \mathcal{A}'(z_k^{\ell-1}). \tag{2.68}$$

It is now apparent that to perform the parameter updates in equation (2.63) and equation (2.62), one must compute a possibly very lengthy chain rule. It may be

the computational physicist's desire to turn to finite-difference methods. However, we stress that not only would this be computationally inefficient, it would also introduce truncation errors which are undesirable and avoidable. The machine learning practitioners' choice of differentiation algorithm is *automatic differentiation* and is explored in literature such as Baydin, *et al.* [Bay+17] (a machine learning-oriented review) and Griewank and Walther [GW08] (a classic textbook on automatic differentiation methods).

Automatic differentiation is more closely related to symbolic differentiation, except a numerical value is returned in the former, and a symbolic expression is returned in the latter. This has an immediate advantage: we are not storing unnecessarily large expressions in computer memory—we only return the numerical values which is exactly what we care about when updating our network parameters. Modern machine learning libraries such as `PyTorch` and `Tensorflow` have automatic differentiation libraries, and it is easy to implement all of the equations of optimisation in just a few lines of code.

2.6 Coupled neural networks

We introduce the *coupled neural network*, a novel framework developed for this thesis to model two or more systems or learning tasks which are interdependent by design. This interdependence arises in scenarios like modelling physical systems with coupled differential equations, where the state of one variable directly influences the other. This framework could be extended to other scenarios where multiple learning tasks or systems exhibit interdependence, such as multi-agent reinforcement learning or modelling complex biological processes. Figure 2.7 demonstrates a coupled neural network, where two separate neural networks share the same input but propagate separate weights and biases throughout the training.[8] The evaluation

[8] Coupled neural networks are different to so-called Siamese neural networks, which share weights and biases across two separate neural networks. In a coupled neural network, the parameters are unique for each model, and therefore each coordinate, but are intrinsically linked in the loss function.

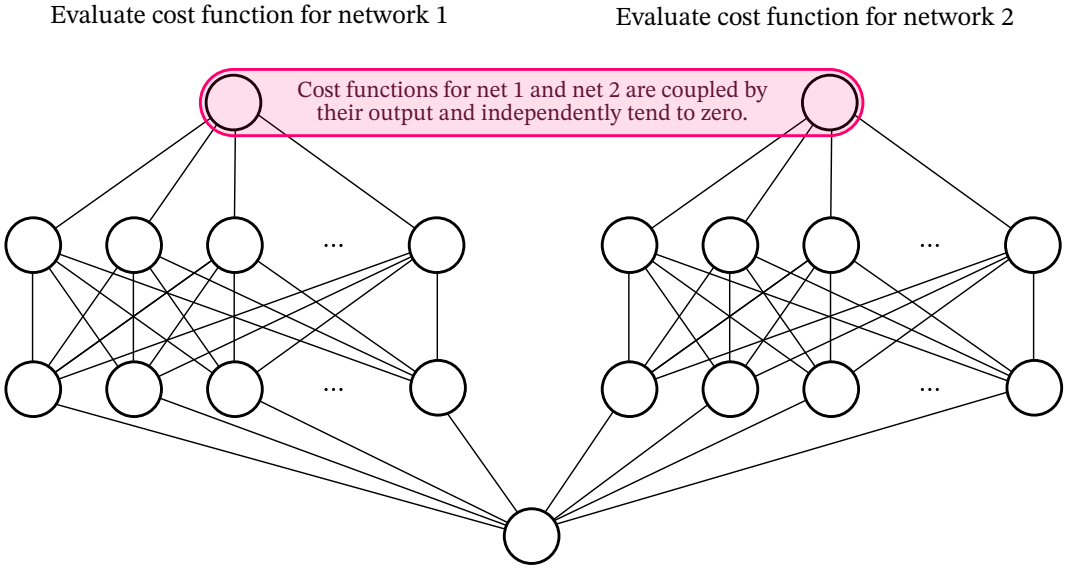


Figure 2.7: A coupled neural network is shown. It propagates independent parameters for each model (given a shared input) and couples only in the evaluation of the cost functions. In principle, n neural networks may be coupled together with a shared input.

of the cost functions is dependent upon the output of the other neural network. In this thesis, we use coupled neural networks to model initial value problems with coupled generalised coordinates with a common time domain, and the coupling is introduced only in the evaluation of an appropriate cost function (which we will discuss further in part III, chapter 4).

We also introduced a coupled optimiser scheme, which ensures that the gradients aren't reset before other coordinates are updated. During backpropagation, the computational graph (which includes the values of the gradients) is preserved in computer memory for all models except the final model; the final model is an arbitrary choice. We backpropagate the final coordinate's cost function and discard its computational graph—the computation of the cost via backpropagation is complete, and no further gradients need to be stored (so computer memory is then freed). The coupled optimiser scheme involves simultaneous optimisers for each coordinate. In principle, the coupled optimiser could include as many optimisers as computer

memory allows. We refer the reader to our code [Gri24c] for further implementation details.

2.7 Convolutional neural networks

2.7.1 Why use a convolutional neural network?

Suppose we want to use an $N \times N$ pixel image as the input to our neural network. We are assuming that the dimensionality of the hidden representation is the same and that every pixel in the input image maps to a pixel in the hidden representation, the number of weights required for this *single* layer-to-layer computation is $(N^2)^2 = N^4$. For an image of the order of megapixels, this is of the order of one trillion (10^{12}) parameters—as of August 2024, no machine learning model is described by so many parameters.[9] The answer to this computational nightmare lies in the *convolutional neural network*.

Convolutional neural networks (ConvNet or CNNs) are designed to detect features in data—typically images or data in matrices—as they pass through the network. Introduced by LeCun, *et al.* [LeC+89; LeC+95a] and influenced by earlier work by Fukushima [Fuk80], CNNs have become a cornerstone in computer vision and other fields dealing with spatial data or other data which is prohibitively large to use using fully connected neural networks.[10]

2.7.2 Components of a convolutional neural network

Convolutional neural networks consist of two components: an image processing pipeline and a prediction/classification pipeline. The first pipeline introduces several techniques to reduce the dimensionality of whatever is input into the network. This

[9] It will be interesting to see how quickly trillion-parameter models appear after the publication of this thesis!

[10] Whilst similar in spirit, the work by Fukushima does not use backpropagation to train neural networks (because it did not exist at that time). LeCun and co-authors were the first group to apply backpropagation to adjust the network parameters automatically. The first implementations of convolutional neural networks on the GPU date from around 2006–2012 [CPS06; UB09; SKP10; Cir+11; KSH12].

is not just for computational convenience. This reduction in dimensionality serves multiple purposes:

- Feature extraction: By applying weights matrices (also known as kernels or features, and we will interchangeably use all three terms in this thesis as appropriate) across the input, the network learns to detect important features such as edges, textures, and more complex patterns. This process abstracts the raw input data into more meaningful representations for the machine learning model to use in the prediction/classification pipeline.
- Translation invariance and local spatial coherence: Through operations like pooling, the network becomes less sensitive to the exact position of features, making it more robust to variations in input data. Pooling summarises small regions (either by taking the maximum or average value in that region) where it is likely that data will be highly correlated in that region.
- Computational efficiency: By reducing the data dimensions, CNNs can process large inputs (like high-resolution images) with far fewer parameters than fully connected networks, making them more computationally tractable.
- Overfitting reduction: The reduced number of parameters and the shared weights across the input space help to prevent overfitting, especially when dealing with high-dimensional data like images.

Figure 2.8 shows a schematic of a convolutional neural network alongside a fully connected neural network to predict whether an input image is a cat or a dog. The first pipeline takes the image, performs a set of operations known as a *convolution* with a set of filters which extract out several features from the input (e.g., the whiskers of a cat or its pointy ears), and pools these features to reduce the dimensions of the image further, for the reasons outlined above. The second pipeline transforms the rich spatial data into data suitable for prediction or classification (to answer questions such as: What is the probability that the image is a cat? How many whiskers does the cat have?)

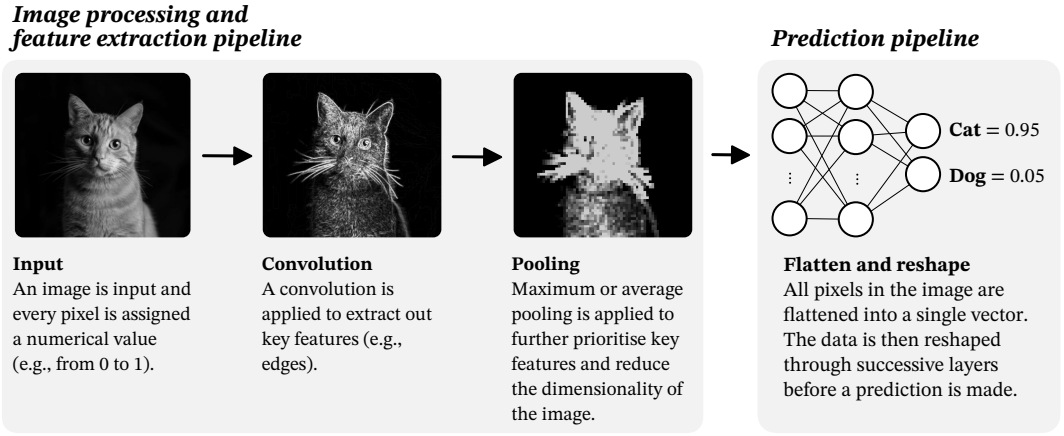


Figure 2.8: An image of a cat is input to a neural network. After a convolutional layer and a pooling layer, the image is flattened and the value of every pixel (from 0 to 1 depending upon how bright the pixel is) is input into the first layer of a feedforward and fully connected neural network. After successive fully connected layers, the network should have sufficient capacity (provided it has been trained) to predict whether the image is of a cat or a dog. [Image '*Charlie*' reproduced with permission from Paul Flannigan Photography [Fla20].]

2.7.3 Definitions of the convolution and cross-correlation operations

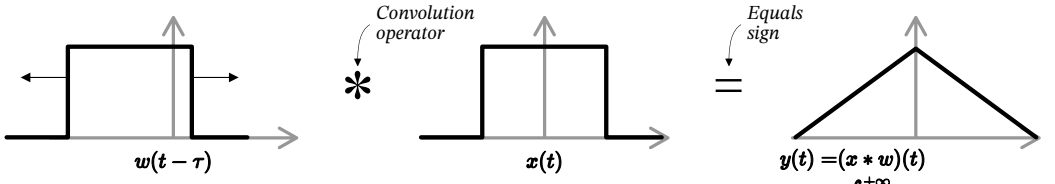
Convolution operation

Let $x, w \in \mathbb{R}^n \to \mathbb{R}$ be two functions which are Lebesgue integrable. The convolution of x and w produces a third function

$$y(t) = (x * w)(t) = \int_{-\infty}^{+\infty} \mathrm{d}t\, x(\tau) \cdot w(t - \tau) \tag{2.69}$$

for some $t \in \mathbb{R}^n$. The function x is the input, w is the kernel and $y : \mathbb{R}^n \to \mathbb{R}$ is the feature map (output). If x and w were discrete functions, i.e., $x, w \in \mathbb{Z}^n \to \mathbb{R}$, then

$$y(t) = (x * w)(t) = \sum_{\tau=-\infty}^{+\infty} x(\tau) \cdot w(t - \tau). \tag{2.70}$$



The parameter τ translates the kernel/window/filter, w, throughout the domain of τ.

The function, x, remains static as w is passed over it.

The convolution of the two functions is obtained by determining the overlap of x and w as x is translated over τ.

Figure 2.9: A kernel (or window or filter), $w(t-\tau)$, passes over the domain, t, by a translation of the adjustable parameter τ. As the kernel moves through the domain, it also passes over the function $x(t)$. The overlap of $x(t)$ and $w(t-\tau)$ is determined through their (for example) Frobenius inner product. The overlap produces a new function, $y(t)$. *Fun fact!* Successive applications of the rectangular function $w(t)$ over the output functions of this convolution result in a normal distribution by the central limit theorem.

The variable τ represents all the shifts of the kernel function across the input function. As τ varies, the kernel function slides across the entire range of the input function. This sliding action is key to convolution: it allows one to 'see' how the kernel function w interacts with the input function x at every point, which is what produces the output function $y(t)$. In a physical sense, if one thinks of x as a signal and w as a filter, then $y(t)$ tells you how much of the signal passes through the filter at every point in time. It may be the case that specific values in the summation are zero, especially when the kernel acts outside of the domain τ.

Figure 2.9 shows the continuous convolution operation over a function, $x(t)$, and a kernel, $w(t)$.

Equation (2.70) is extended to matrices by introducing another index, σ,

$$\mathsf{Y}(s,t) = (\mathsf{X} * \mathsf{W})(s,t) = \sum_{\sigma=-\infty}^{+\infty} \sum_{\tau=-\infty}^{+\infty} \mathsf{X}(\sigma,\tau) \cdot \mathsf{W}(s-\sigma, t-\tau), \tag{2.71}$$

where we use the parentheses (s,t) to denote the matrix element at position s and t.

Convolutions are commutative,

$$u * v = v * u, \tag{2.72}$$

associative,

$$(u * v) * w = u * (v * w), \tag{2.73}$$

and linear,

$$\begin{aligned} (ax + bx') * w &= ax * w + bx' * w \\ x * (aw + bw') &= ax * w + bx * w'. \end{aligned} \tag{2.74}$$

Because convolutions are linear, we will also require (non-linear) activation functions when introducing these operations to our neural network (the argument is the same as before: a sequence of linear transformations is the same as just one linear transformation).

Cross-correlation operation

Closely related to the convolution operation is the cross-correlation function

$$y(t) = (x \star w)(t) = \sum_{\tau=-\infty}^{+\infty} x(t+\tau)w(\tau), \tag{2.75}$$

which is effectively a clockwise rotation of the kernel by 180 degrees. Note the sign change in the argument of w, compared with equation (2.70), and the change of operation from $*$ to $\star$. Equation (2.75) is extended to matrices by introducing another index, σ,

$$\mathsf{Y}(s,t) = (\mathsf{X} \star \mathsf{W})(s,t) = \sum_{\sigma=-\infty}^{+\infty} \sum_{\tau=-\infty}^{+\infty} \mathsf{X}(s+\sigma, t+\tau) \cdot \mathsf{W}(\sigma,\tau). \tag{2.76}$$

Appendix D demonstrates that the cross-correlation operation may be interpreted as a Frobenius inner product of a sub-matrix of X and the kernel, W.

For historical and conventional reasons, it is actually the cross-correlation that is calculated in the leading machine learning libraries such as PyTorch [PyT24] and Tensorflow [Ten24]—'convolutional' neural networks are a technical misnomer, but during training, convolutions and cross-correlations should converge on the same result.

Since most of the data we are interested are two-dimensional matrices, it is equation (2.76) that we will focus on in this section and in part III, chapter 6 of this thesis.

2.7.4 Image processing and feature extraction pipeline

Features, filters and their initialisation

We need to initialise a weights matrix (filter) for every feature that we wish to extract in a given layer. The number of features to be extracted in a given layer is not known *a priori* and must be adjusted post-training as appropriate. Let the N weights matrices in layer ℓ be W_i^ℓ, where $i \in \{1, \cdots, N\}$. Let an element of the weights matrices be $\mathsf{W}_i^\ell(s,t)$.

The elements of the weights matrix in a convolutional neural network are initialised from the uniform distribution [PyT24]

$$\mathsf{W}_i^\ell(s,t) \sim U\left(-\sqrt{\frac{1}{F_{\ell-1}k_W^2}}, \sqrt{\frac{1}{F_{\ell-1}k_W^2}}\right), \tag{2.77}$$

where $F_{\ell-1}$ are the number of features in the previous layer, $\ell - 1$, and k_W is the width of the weights matrix.[11] Biases are also drawn from this distribution.

The weights matrices filter out important features such as edges or curvature in data which is passed through them. The goal is still to find the appropriate weights and biases which describe the mapping between the input data and the output data.

[11] The number of features to be learnt is associated with a 'channel'. For the input layer, if you have a colour image, then you send red, green and blue colour channels through the neural network. These three channels may also be thought of as features. If the input is just a greyscale image, then this is a single channel or feature. We do a similar thing for the features extracted at any other layer of the CNN.

To preserve the input dimensions after the convolution or cross-correlation operation, we apply a padding of zeros around the input image or feature map. For a weights matrix (kernel) of size $k_W \times k_W$, where k_W is odd, we add κ zeros to each side of the input, where

$$\kappa = \frac{k_W - 1}{2}. \tag{2.78}$$

This padding ensures that the convolution operation can be applied to every pixel of the original input without reducing the spatial dimensions of the output at this stage.[12]

Step 1: determine the cross-correlation of each feature with the weights matrices

The cross-correlations of the i features input to a layer, $\mathsf{Y}_i^{\ell-1}$, with the weights matrices, W_i^{ℓ}, are

$$\mathsf{Z}_i^{\ell}(s,t) = (\mathsf{Y}_i^{\ell-1} \star \mathsf{W}_i^{\ell})(s,t) = \sum_{\sigma=-\kappa}^{\kappa} \sum_{\tau=-\kappa}^{\kappa} \mathsf{Y}_i^{\ell-1}(s+\sigma, t+\tau) \cdot \mathsf{W}_i^{\ell}(\sigma,\tau) \tag{2.79}$$

Figure 2.10 demonstrates the cross-correlation of four weights matrices to produce four feature maps from the cat classification problem described previously.

Step 2: pool the resultant features

Pooling reduces the spatial dimensions of input data and, along with sparse connections and weight sharing, further reduces the number of parameters in the neural network. It is computationally expensive and redundant to explore every pixel of an image, since neighbouring pixels are extremely likely to be similar. Pooling operations summarise data in small regions into a single value using a kernel of size $k_P \times k_P$.[13] Average pooling (AvgPool) computes the average value in that region, giving a general representation of that region, whereas maximum pooling (MaxPool)

[12] Kernels almost always have odd dimensions because we centre the kernel on a given pixel. Asymmetric kernels are highly unusual.

[13] The size of the pooling kernel, k_P^2 need not be the same as the size of the weights matrices, k_W^2.

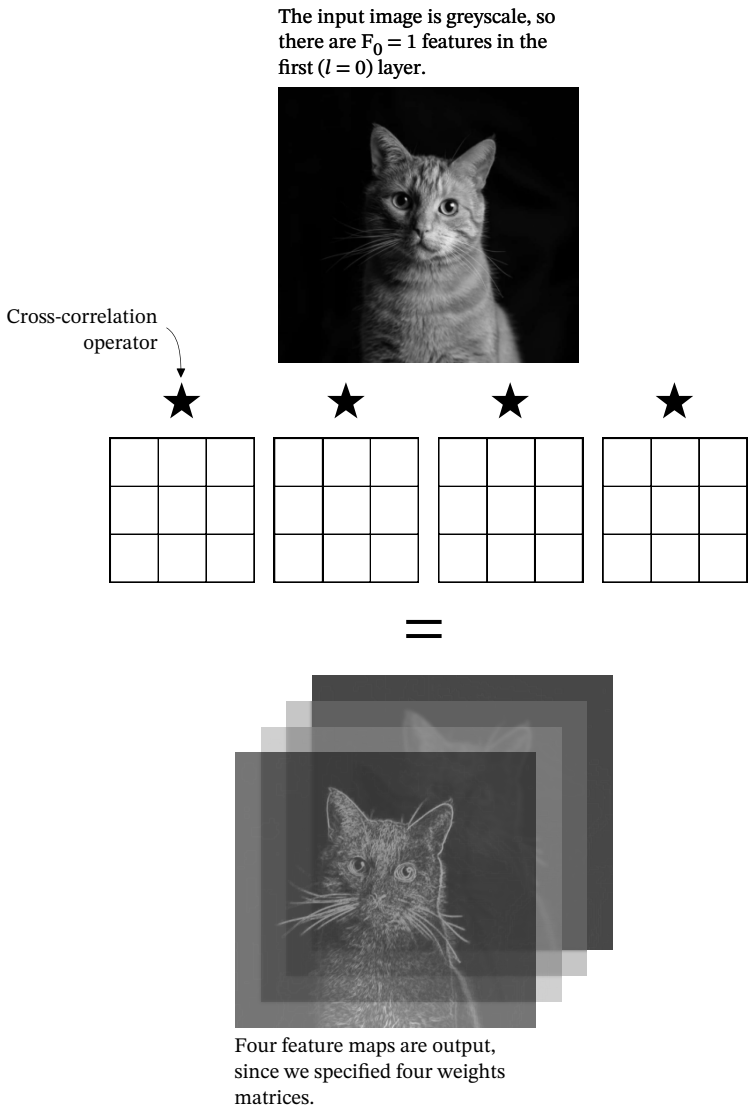


Figure 2.10: An image of a cat is input to a neural network. We determine the cross-correlation of the input image with four weights matrices (filters) to produce four features.

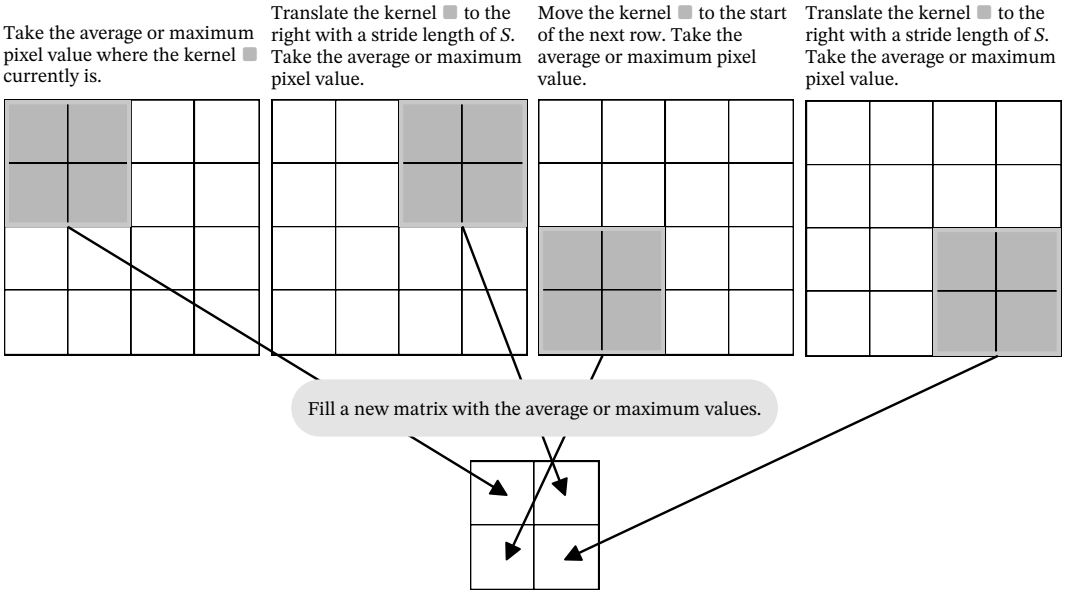


Figure 2.11: Average or maximum pooling using a kernel of size $k_P \times k_P = 2 \times 2$ and stride length $\Sigma = 2$ on a 4×4 grid. The output is a square matrix with reduced width and height $\lfloor (N - k_P)/\Sigma \rfloor + 1 = \lfloor (4 - 2)/2 \rfloor + 1 = 2$.

retains the maximum value, emphasising the most prominent feature. More aggressive pooling and reduction of the input dimensions can be performed by increasing the stride, Σ, which is the number of pixels the kernel should be translated by in all dimensions. The width or height of the output of a pooling operation is given by

$$\left\lfloor \frac{N - k_P}{\Sigma} \right\rfloor + 1. \tag{2.80}$$

Maximum pooling is given by

$$\mathsf{P}^{\ell}(s,t) = \max_{0 \le \sigma < k_P, 0 \le \tau < k_P} \mathsf{Z}^{\ell}(\Sigma s + \sigma, \Sigma t + \tau). \tag{2.81}$$

Average pooling is given by

$$\mathsf{P}^{\ell}(s,t) = \frac{1}{k_P^2} \sum_{\sigma=0}^{k_P-1} \sum_{\tau=0}^{k_P-1} \mathsf{Z}^{\ell}(\Sigma s + \sigma, \Sigma t + \tau). \tag{2.82}$$

Figure 2.11 demonstrates either average or maximum pooling.

Pooling necessarily introduces translation invariance in the detection of features (for example, when detecting a human face, the machine learning model does not care where the eyes are in relation to the nose).[14]

Suppose one is looking for the presence of a vortex in an atomic density profile—if the vortex shifts a little between observations, maximum pooling would still detect the vortex because it looks for the strongest signal within each patch, not the precise location.[15]

Step 3: apply a bias and then an activation function

After determining the cross-correlations of the layer input with the set of weights matrices, and before applying the activation function we apply a bias b_i^ℓ element-wise

$$\mathsf{P}^\ell(s,t) \to \mathsf{P}^\ell(s,t) + b_i^\ell \mathbb{1}_{p\times p}, \tag{2.83}$$

where, assuming that $\mathsf{P} \in \mathbb{R}^{p\times p}$, $\mathbb{1}_{p\times p}$ is a matrix of ones of size $p \times p$.

After every convolutional operation, a non-linearity must be applied to ensure non-linear mappings may be learnt (which is essentially always the case for scenarios when a convolutional network is required).[16] The output of a convolutional layer is then

$$\mathsf{Y}^\ell(s,t) = \mathcal{A}\big(\mathsf{P}^\ell(s,t)\big). \tag{2.84}$$

[14] Hinton, *et al.* [HKW11], suggest an alternate (so-called *capsule*) architecture where pooling is not used; instead, the position, orientation and scale of an object are deemed to be more important than spatial invariance [SFH17]. Translation invariance means that if the input image is shifted slightly, the output of the pooling layer doesn't change much. Pooling also exploits local spatial coherence—nearby pixels or data points are expected to be similar.

[15] The work by Metz, *et al.* [Met+21, op. cit.], detects vortex positions in atomic density samples. Whilst they do not comment on the role of maximum pooling in their network, translation invariance is, indeed, being introduced by the presence of these operations.

[16] Some literature suggests that the pooling operation should be applied after the non-linearity [Bro19], but we note that if the non-linearity is monotonically increasing (e.g., a ReLU activation for $x > 0$), then the pooling operation and non-linearity are commutative. In any case, we argue that the ordering of operations is largely unimportant since most of the compute time is taken performing the convolution operation, rather than the application of a non-linearity (indeed, we observe no noticeable improvement in training time by swapping the order of operations in our experiments in part III, chapter 6).

We therefore have the convolutional neural network analogue[17] of the layer outputs:

$$\mathsf{Y}^{\ell}(s,t) = \mathcal{A}\big(\mathsf{P}^{\ell}(s,t)\big) \quad \text{where} \tag{2.86a}$$

$$\mathsf{P}^{\ell}(s,t) = \text{Pooling}\big[\big(\mathsf{Y}^{\ell-1} \bigstar \mathsf{W}^{\ell}\big)(s,t)\big]. \tag{2.86b}$$

2.7.5 Prediction and classification pipeline

After the application of successive convolutional and pooling layers, and the extraction of appropriate features, classification of the input data can begin. Machines do not care about the structure of the input data; to a computer, a $n \times n$ matrix is equivalent to a one-dimensional vector of size n^2, so long as they contain the same values. We are therefore at liberty to flatten the result of the convolutional section of our neural network from a matrix into a vector. Elements of this vector are then passed through a feedforward and fully connected neural network, exactly as previously presented in chapter 2.

After flattening the output of the convolutional layers into a vector, we pass this through a series of fully connected layers. For the ℓ-th layer in this fully connected section, the output of the j-th neuron is given by equations (2.4a–2.4b).

If the output layer, L, is a set of probabilities, we typically use a softmax activation to get probabilities for each class,

$$y_j^L = \frac{e^{z_j^L}}{\sum_{i=1}^{n_L} e^{z_i^L}}, \tag{2.87}$$

[17] In one dimension, it is clear to see the equivalency between a fully connected neural network in equations (2.4a–2.4b).

$$y_j^{\ell} = \mathcal{A}(z_j^{\ell}) \quad \text{where} \tag{2.85a}$$

$$z_j^{\ell} = \sum_{k=1}^{n_{\ell-1}} y_k^{\ell-1} \star w_{jk}^{\ell} + b_j^{\ell}, \tag{2.85b}$$

where we have omitted pooling.

where n_L is the number of neurons in the output layer (typically equal to the number of classes).

If the output layer is something else (e.g., the number of vortices in a Bose–Einstein condensate, or its temperature, etc.), then an activation function is typically not used in the final layer.

This fully connected section of the network learns to map the high-level features extracted by the convolutional layers to the final class probabilities, classifications, or any other desired outcome of the model.

An appropriate cost function (e.g., the mean square error of the expected output with the neural network's output) is defined and then the entire network, including both the convolutional and fully connected parts, is trained end-to-end using an appropriate learning algorithm (see § 2.5) with scalar-valued backpropagation (see § 2.5.4), allowing it to learn both the feature extraction and classification tasks simultaneously.

2.7.6 Computational considerations

In machine learning libraries, data is stored in multi-dimensional arrays. Features are stacked together, occupying separate channels within this multi-dimensional array. This means that each layer represents a multi-dimensional array of size $F_\ell \times N \times N$, where F_ℓ are the number of features in that layer.

Furthermore, we typically use batching (see § 2.5.1) in convolutional neural networks, whereby β samples are randomly selected, without replacement, and are passed through the CNN together, sharing the same weights matrices. Smaller batch sizes (e.g., around 16 or 32 inputs per batch) generally lead to better generalisation [Kes+17] by introducing more stochasticity in the optimisation algorithms. This helps the model escape shallow local minima and find more robust solutions that generalise better to unseen data. We add the batches of images as another dimension to the multi-dimensional array, such that each layer is $\beta \times F_\ell \times N \times N$.

In the machine learning literature, multidimensional arrays are referred to as 'tensors'. These are not tensors in the literal mathematical sense (they do not transform like a tensor), nor can they be interpreted as elements of tensor products.

2.8 Complex-valued neural networks

Almost all of our analysis thus far has focussed on assuming that the neuron inputs and weights are real-valued. One may have noticed that we briefly alluded to the existence of *complex*-valued neural networks when we discussed AdaGrad and other optimisation algorithms. A wave of excitement can be seen on a physicist's face when they realise that they can represent certain physical phenomena (such as waves, potential flow in two dimensions, many aspects of quantum mechanics, and if they are of the experimental or engineering persuasion—AC circuits) using complex numbers. A natural question to ask is what, if anything, lies at the intersection of neural networks and complex analysis?

This section requires knowledge of basic complex analysis, which is covered in appendix (V).

2.8.1 Fully connected complex-valued neural network

The activation of a neuron in a fully connected neural network is

$$z_j^\ell = \sum_{k=1}^{n_{\ell-1}} w_{jk}^\ell y_k^{\ell-1} \tag{2.88a}$$

$$y_j^\ell = \mathcal{A}\big(z_j^\ell\big), \tag{2.88b}$$

where $z_j^\ell, y_j^\ell \in \mathbb{C}$ are the pre- and post-activation neuron values and $w_{jk}^\ell \in \mathbb{C}$ are weights. For now, we will omit the bias, b_j^ℓ, which is typically taken to only have a real component.

2.8.2 Activation functions in the complex domain

Early machine learning literature suggested the use of bounded, monotonic activation functions such as the sigmoid or hyperbolic tangent. Liouville's theorem in complex analysis says that every bounded and entire function must be constant [ST18]. As a consequence of Liouville's theorem, activation functions in the complex domain must be non-holomorphic. Non-holomorphic functions are introduced at least by i) using complex-valued parameters in the cost function and ii) using non-holomorphic activation functions [Hir13]. Complex-valued activation functions may treat the real and imaginary parts of the neuron inputs separately,

$$\mathcal{A}(z) = \mathcal{A}(\Re z) + i\mathcal{A}(\Im z), \tag{2.89}$$

where $\mathcal{A}$ may be a hyperbolic tangent, ReLU, or any other appropriate activation function on the real domain. Complex-valued activation functions may otherwise treat the phase and amplitude of the neuron inputs separately,

$$\mathcal{A}(z) = f(|z|)\,\mathrm{e}^{\arg z}, \tag{2.90}$$

where $f : \mathbb{R} \to \mathbb{R}$.

We can also derive complex-valued probabilistic activations (following the discussion in § 2.2.5). A complex random variable $Z = X + iY$ is described by a pair of real random variables X and Y. The complex random variable is described by a joint distribution of its real and imaginary parts, $p(z) = p(x, y)$. The cumulative probability distribution, $\phi(z) = \phi(x, y)$, is defined via this joint distribution. Any complex-valued probabilistic activation function is defined as

$$\mathcal{A}(z) = z\phi(z) = z \int_{-\infty}^{x} \int_{-\infty}^{y} p_{X,Y}(u, v)\,\mathrm{d}u\mathrm{d}v, \tag{2.91}$$

where we have introduced subscripts on $p_{X,Y}(u, v)$ for clarity; we have introduced the dummy variables u and v.

We provide a derivation of the modReLU activation function (an amplitude-phase function) and derive a new activation function which we call *cGELU* (a real-imaginary function).

ModReLU

Consider the complex-valued Dirac delta distribution [HS98]

$$\delta(z) = \delta(x, y) = \delta(x + iy) = \delta(x)\delta(y). \tag{2.92}$$

The cumulative distribution for the 2-dimensional Dirac delta distribution is the product of two Heaviside functions

$$\phi(z) = \int_{-\infty}^{x}\int_{-\infty}^{y} \delta(u)\delta(v)\,\mathrm{d}u\mathrm{d}v = \Theta(x)\Theta(y). \tag{2.93}$$

The complex-valued probabilistic activation function is thus

$$\mathcal{A}(z) = z\phi(z) = z\Theta(x)\Theta(y) \equiv \max(0, |z|)\mathrm{e}^{\arg z}. \tag{2.94}$$

Whilst not derived in this way in the literature, this is referred to as an unbiased ($b = 0$) modReLU function [ASB15] (Arjovsky, *et al.*, simply state modReLU based on intuition—it is not a function which has been derived). A trainable bias term is commonly added; the bias acts as a threshold for the activation of the neuron. This activation function is considered to be the complex analogue of the ReLU activation function,

$$\mathrm{modReLU}(z) = \max(0, |z| - b)\mathrm{e}^{\arg z}. \tag{2.95}$$

Figure 2.12 shows the unbiased modReLU function. Notably, it is magnitude- and phase-preserving only in the quadrant $\Re z > 0, \Im z > 0$, analogous to ReLU in the real domain.

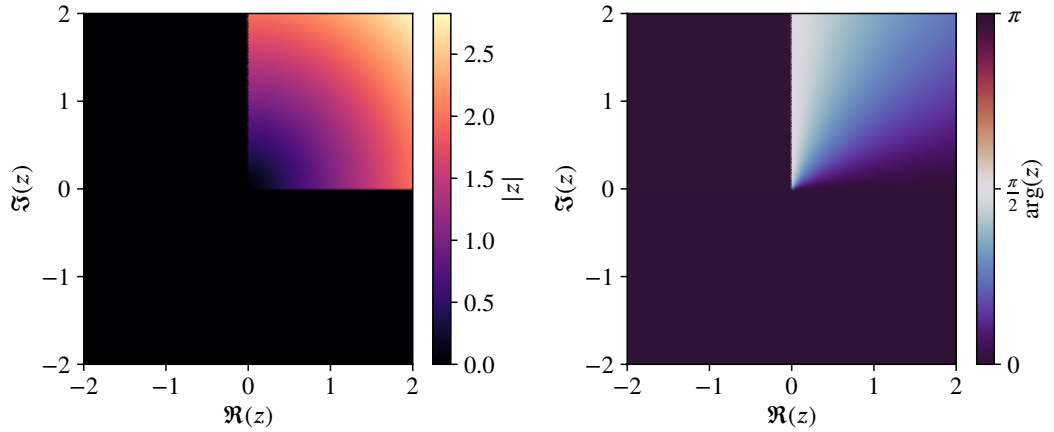


Figure 2.12: The magnitude (left) and phase (right) of a complex number z passed through the activation function $\text{modReLU}(z)$ is shown. modReLU is magnitude- and phase-preserving only in the upper right quadrant of the complex plane.

cGELU

We now introduce cGELU, an activation function which is phase-preserving everywhere in the complex plane. Consider again the complex random variable $Z = X + iY$, where X and Y are jointly normal, real-valued, random variables representing the real and imaginary parts of Z, respectively. We will further assume that X and Y are independent and identically distributed real-valued normal random variables with zero mean and variance $\sigma^2 = 1/2$ [Lap17], i.e.,

$$Z \sim \mathcal{CN}(0,1) \iff \Re(Z) \sim \mathcal{N}\left(0, \frac{1}{2}\right) \quad \text{and} \quad \Im(Z) \sim \mathcal{N}\left(0, \frac{1}{2}\right), \tag{2.96}$$

where $\mathcal{CN}(0,1)$ is a standard complex normal distribution. Since we assumed that X and Y are independent, their joint probability density function is the product of their individual probability density functions

$$p_{X,Y}(x,y) = p_X(x)p_Y(y) = \frac{1}{\sqrt{\pi}}\exp(-x^2) \times \frac{1}{\sqrt{\pi}}\exp(-y^2) = \frac{1}{\pi}\exp(-x^2 - y^2). \tag{2.97}$$

The cumulative distribution function is

$$\phi_Z(x, y) = \int_{-\infty}^{y} \int_{-\infty}^{x} \frac{1}{\pi} \exp(-u^2 - v^2) \, \mathrm{d}u\mathrm{d}v. \tag{2.98}$$

Since X and Y are independent, we can separate the double integral into the product of two single integrals

$$\phi_Z(x, y) = \left(\int_{-\infty}^{x} \frac{1}{\sqrt{\pi}} \exp(-u^2) \, \mathrm{d}u \right) \times \left(\int_{-\infty}^{y} \frac{1}{\sqrt{\pi}} \exp(-v^2) \, \mathrm{d}v \right). \tag{2.99}$$

Each of these single integrals represents the cumulative distribution function of a standard real normal distribution with zero mean and variance 1/2, which can be expressed as

$$\phi_Z(x, y) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \times \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{y}{\sqrt{2}}\right)\right]. \tag{2.100}$$

The cGELU activation function is then defined as the product of the input z and the cumulative distribution function of the standard complex normal distribution evaluated at the real and imaginary parts of z:

$$\mathrm{cGELU}(z) = z\phi_Z(x, y) = (x + iy) \cdot \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \cdot \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{y}{\sqrt{2}}\right)\right]. \tag{2.101}$$

This definition preserves the phase information of the input, as it multiplies the complex input z by a real-valued factor $F_Z(x, y)$.

Figure 2.13 shows the cGELU function. In contrast to modReLU, it is phase-preserving everywhere and magnitude-preserving in and just outside of the quadrant $\Re z > 0, \Im z > 0$.

2.8.3 Optimisation in the complex domain

Recall the role of the cost function in the real domain, as discussed in § 2.5, as a metric used to find the optimal parameterisation (of weights and biases) of a neural network. In a complex-valued neural network, the network parameters are complex-

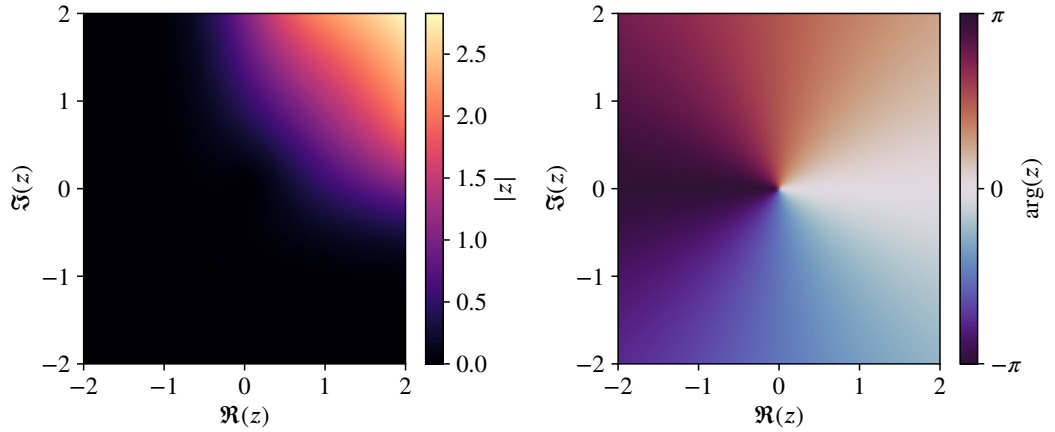


Figure 2.13: The magnitude (left) and phase (right) of a complex number z passed through the activation function cGELU(z) is shown. cGELU is magnitude-preserving only in the upper right quadrant of the complex plane but phase-preserving everywhere.

valued weights and real-valued biases. The complex domain lacks the concept of ordering, which is necessary for any gradient-based optimisation routines. The cost function must continue to map to the real domain, $\mathcal{C} : \mathbb{C} \to \mathbb{R}$. Using the Wirtinger calculus in appendix A.2, it is a trivial task to describe the update procedure for the complex-valued parameters to give a real-valued loss function.

First, we split the complex-valued weights, w_{jk}^{ℓ}, into its real and imaginary parts, $\Re w_{jk}^{\ell}$ and $\Im w_{jk}^{\ell}$, respectively. We will apply a gradient descent to the real and imaginary parts using a step size $\eta/2$. The update to the real and imaginary parts are

$$\Delta \Re w_{jk}^{\ell} = -\frac{\eta}{2} \frac{\partial C}{\partial \Re w_{jk}^{\ell}} \tag{2.102a}$$

$$\Delta \Im w_{jk}^{\ell} = -\frac{\eta}{2} \frac{\partial C}{\partial \Im w_{jk}^{\ell}}. \tag{2.102b}$$

Combining the real and imaginary parts into a single, complex update procedure gives

$$\Delta w_{jk}^{\ell} = -\frac{\eta}{2}\frac{\partial C}{\partial \Re w_{jk}^{\ell}} - i\frac{\eta}{2}\frac{\partial C}{\partial \Im w_{jk}^{\ell}} \tag{2.103}$$

$$= -\frac{\eta}{2}\left(\frac{\partial C}{\partial \Re w_{jk}^{\ell}} + i\frac{\partial C}{\partial \Im w_{jk}^{\ell}}\right) \tag{2.104}$$

$$\Delta w_{jk}^{\ell} = -\eta\frac{\partial \mathcal{C}}{\partial (w_{jk}^{\ell})^{*}}, \tag{2.105}$$

where in the last line we have used equation (A.12). This states that the update procedure for complex weights *only* considers the conjugate of the weights in the update procedure.

2.9 Training, validating and testing models

2.9.1 Training, validation, and testing datasets

In most supervised learning tasks, the training and evaluation of the machine learning model takes place over three distinct datasets: the training, validation, and testing datasets. The training dataset is used to learn the underlying patterns and structures in the data—it is used to build the model. The validation dataset is used to test the model's performance throughout the training and is used to inform decisions about, for example, when to stop training (to avoid overfitting). The validation dataset is also used to select the best model. The test dataset then provides an unbiased evaluation of the chosen model's performance; the test data is never seen by the model during training, and so may be regarded as a 'real world' demonstration of the model.

In most cases, data is randomised and split into 80% training, 10% validation, and 10% testing. Some literature suggests performing a stratified sample to avoid clustering similar data together in a particular dataset, which a truly random sample might do.

2.9.2 Stochasticity in machine learning

Machine learning models are inherently random. Randomness may be a result of human input to the model. How was the training data obtained, and how was it subsampled if there was too much data? As previously discussed, randomness is a part of the initialisation of the machine learning model. Randomness may also be introduced by the order in which machine learning models observe data from the training dataset, as discussed in § 2.5.1; and the way in which we split our dataset into training, validation, and testing datasets, as discussed in § 2.9.1. Machine learning models exhibit run-to-run variation, give different outcomes (e.g., classifications, solutions to differential equations) when used, and have different performance characteristics (see § 2.9.3). All of these differences are, however, within a range. We therefore say that machine learning models are *stochastic*.

2.9.3 When is a model 'good'?

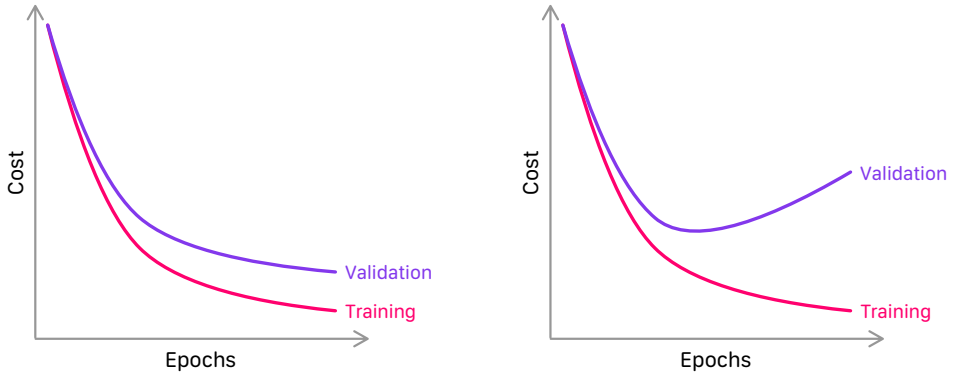


Figure 2.14: This schematic shows trajectories that the cost function may take as training and validation progresses. (a) shows 'good' training and validation behaviour—both metrics are decreasing. (b) shows 'bad' training and validation behaviour—it demonstrates overfitting since the validation metric is diverging significantly from the training metric.

There are many strategies to determine whether a machine learning model is appropriately learning the underlying structures and patterns in data. Corrective strategies also exist to fix issues that may arise when training a model.

Figure 2.14 demonstrates two scenarios which are typically seen in classification problems. When building a model, the training and validation cost metrics are obtained throughout the training procedure. If both metrics are almost always decreasing, then the modelling is behaving as expected and is not overfitting to the training data. If the validation cost metric diverges significantly from the training cost metric, then this can be interpreted as overfitting. The fixes for overfitting are generally simple to implement: i) early stopping can be used to terminate training at the point where overfitting begins, ii) more data can be collected and input through the network, and iii) simpler models with fewer model parameters to reduce the network capacity.

The stochastic nature of optimisation algorithms can give rise to spikes in the cost functions—this is completely normal and does not mean that the network is overfitting.

2.10 Summary

We have presented an overview of the mathematics of machine learning for the physicist. By developing a consistent nomenclature in this chapter, we have explored the fully connected and feedforward neural network—foundational to all modelling in this thesis. We motivated the need for probabilistic activation functions and developed the coupled neural network, both of which will be essential in our discussion of neural initial value and eigenvalue problems. We discussed the convolutional neural network, necessary in all image recognition and classification tasks. We finally discussed complex-valued neural networks, which our work briefly explored, and applied our probabilistic activation function framework to functions in the complex domain.

Pseudocode for a minimal example of neural network is listed in algorithm 1.

Algorithm 1 Minimal Neural Network Example

```
Load data and labels
Split datasets into train, validation, and test sets
Choose model architecture (e.g., convolutional, coupled neural network)
Initialise network parameters (weights and biases)
Define hyperparameters (learning rate, epochs, batch size)
Select optimisation algorithm (e.g., Adam)
for epoch = 1 to num_epochs do
    for each batch in training_data do
        Forward pass: compute predictions
        Calculate loss
        Backward pass: compute gradients
        Update weights and biases using chosen optimiser
    end for
    Evaluate model performance on validation set
end for
Test model on held-out test set
```

In part III, chapters 4 and 5, we will use the fully connected and feedforward neural network to solve initial value and eigenvalue problems. In part III, chapter 6, we will use convolutional neural networks to build a predictive model for the thermodynamical parameters of a Bose gas with a thermal component.

3

BOSE–EINSTEIN CONDENSATION

ARISING from statistical mechanics, Bose–Einstein condensation describes the condensation of a macroscopic collection of particles into the lowest-energy ground state. Bose–Einstein condensation is an interplay between the temperature and density of particles. Whilst experimentally demonstrated only in low-temperature (and low-density) regimes, theories of Bose–Einstein condensation in the interior of neutron stars—a high-temperature and high-density regime—have emerged recently [HBG00; CH12; SCB14; Cha+22; Ind+24].

The first experimental realisation of a Bose–Einstein condensate in a vapour of rubidium-87 atoms was reported in 1995 by Anderson, *et al.* [And+95] around 70 years after their theoretical prediction by Satyendra Bose and Albert Einstein. Later in 1995, the second realisation of a Bose–Einstein condensate was reported in sodium-23 by Davis, *et al.* [Dav+95]. Indeed, Bose–Einstein condensation is observed in a wide range of atomic samples from the alkali, alkali earth and lanthanide series [Bra+95; Mod+01; Ste+09; Lu+11; Aik+12]. The standard isotopes used in weakly interacting atomic experiments include sodium-23 and rubidium-87, which have repulsive atomic interactions at low temperatures.

In this chapter, we will introduce two descriptions of the dynamics of a Bose gas: one in the absence of a thermal component, and one in the presence of a thermal component. The former can be described using the so-called *Gross–Pitaevskii*

equation (introduced in § 3.3) and the latter can be described using the *stochastic Gross–Pitaevskii equation* (introduced in § 3.4). A secondary aim of this chapter is to describe the full numerical implementation of the stochastic Gross–Pitaevskii equation—alongside open source code written in the *Rust* programming language—to assist researchers who may be using this methodology for the first time. We will describe some important computational considerations which lie at the intersection of physics and numerics. This chapter is a foundation for generating the training data for our thermodynamical parameter prediction model in chapter 6.

3.1 Why study Bose–Einstein condensation?

The Bose–Einstein condensate provides a suitable framework for novel technologies such as rotational sensors, interferometers, and atom analogues of electronic components, amongst other applications. Given an ultracold gas of atoms (the gas must be cooled to temperatures of the order of nanokelvins in order to access a quantum-degenerate regime [Ket02]), an appropriate trapping potential, and mechanisms to control atomic flow, one can theoretically and experimentally realise many such technologies. The study of guided, coherent atom-matter wave applications is called *atomtronics* [Ami+21].

One notable application of the Bose–Einstein condensate is in the development of ultra-sensitive rotational sensors [Ryu+13; Wri+13; MOS15; RSB20]. By exploiting the response of a Bose–Einstein condensate to rotational forces, recent experiments have demonstrated prototype gyroscopes and accelerometers which could be used for high-precision measurements. These devices have significant implications for navigation, geophysics, and fundamental physics research [Ami+21, op. cit.].

Bose–Einstein condensates may also be used for atom-based interferometers [CSP09]. Atom interferometers perform precise measurements of gravitational fields, fundamental constants, and other physical quantities [Dim+07], making them ideal for high-precision measurements [Ber+13].

Whilst the field of atomtronics now encompasses all applications of guided atom-matter waves, the term's origin lies in atom-based analogues of electronic components [Mic+04; Pep+09; Sea+07].

The diverse applications of Bose–Einstein condensates are realised by an interplay of theoretical and experimental physics. To faithfully reproduce experimental conditions numerically, we require accurate knowledge of the chemical potential and temperature of the system [DMB01; RBB12]. The chemical potential determines the equilibrium distribution of atoms in the condensate and the thermal cloud, while the temperature governs the extent of thermal fluctuations.

3.2 Thermodynamical picture

We will first approach the theoretical description of Bose–Einstein condensation as it arises in thermodynamics and statistical mechanics. We will then consider a field theoretical approach to derive the equations which govern condensate dynamics.

Let us first consider the grand canonical ensemble—an idealistic construction used to explore the statistics of a system in thermodynamical equilibrium with a reservoir. The probability of realising a configuration of N particles in a distinct microstate with energy E is

$$P(E) = \mathrm{e}^{\beta(\Omega+\mu N-E)}, \tag{3.1}$$

where $\beta = 1/k_B T$ is the thermodynamical beta, k_B is the Boltzmann constant, and T is the absolute temperature. The quantity μ is the chemical potential and is heuristically the energy required to add or remove atoms from the system (the chemical potential is equivalently the mean particle number for a given scattering length). The grand potential, Ω, describes all the relevant thermodynamical parameters for our system

$$\Omega = E - TS - \mu N, \tag{3.2}$$

where S is the entropy of the system.

For a non-interacting system of N particles, the system Hamiltonian $\hat{H}$ is the sum of the independent particle Hamiltonians $\{\hat{H}_i\}$ where each independent Hamiltonian obeys the time-independent Schrödinger equation $\hat{H}_i\psi_i(\mathbf{x}) = \varepsilon_i\psi_i(\mathbf{x})$, where ε_i is the particle eigenenergy and $\psi_i(\mathbf{x})$ is an eigenvector. For a system with a large amount of atoms, solving the Schrödinger equation for each atom is an intractable task! One can introduce the *partition function* Z to describe the ensemble of particles. The partition function also obeys

$$\Omega = -\frac{1}{\beta}\ln Z \tag{3.3}$$

where

$$Z = \prod_{j=0}^{\infty} z_j = \prod_{j=0}^{\infty} \underbrace{\sum_{n_j=0}^{\infty} \mathrm{e}^{\beta n_j(\mu-\varepsilon_j)}}_{\equiv z_j}. \tag{3.4}$$

Noting that z_j is a geometric series over the states n_j, and that $\mathrm{e}^{\beta(\mu-\varepsilon_j)} < 1$, one can write

$$z_j = \frac{1}{1-\mathrm{e}^{\beta(\mu-\varepsilon_j)}}. \tag{3.5}$$

Using equation (3.3), one can show that the grand potential is

$$\Omega = \frac{1}{\beta}\sum_{i=0}^{N-1} \ln\left(1-\mathrm{e}^{\beta(\mu-\varepsilon_i)}\right), \tag{3.6}$$

where the product of logarithms has been simplified, and the summations have been re-indexed to N atoms.

From equation (3.2), it is clear that each thermodynamical quantity can be represented in terms of derivatives of Ω. It is particularly interesting to look at each state's average number of particles. Taking derivatives of (3.2), one can show that

$$N = -\frac{\partial\Omega}{\partial\mu} = \sum_{i=0}^{N-1} \frac{1}{\mathrm{e}^{\beta(\varepsilon_i-\mu)}-1}. \tag{3.7}$$

Noting that this is just the summation over each *state*, one obtains the average state occupation number

$$\bar{n}_i = \frac{1}{\mathrm{e}^{\beta(\varepsilon_i - \mu)} - 1}. \tag{3.8}$$

Equation (3.8) is the *Bose–Einstein statistical distribution*. We shall hereafter refer to the Bose–Einstein distribution as n_{BE}.

The typical thermodynamical picture of a Bose–Einstein condensate system is to consider a reservoir, hereafter referred to as a 'thermal cloud,' at temperature T and with a potential energy to add or remove atoms from the thermal cloud—the chemical potential, μ. Between the thermal cloud and the system, there is a continuous exchange of energy and atoms (where the total energy and the total atom number are conserved quantities between the thermal cloud and the system). The total atom number in equation (3.7) is a summation of the contributions from the *condensate* (mode $i = 0$) with atom number N_0 and the *thermal cloud (non-condensate)* (modes $i > 0$) with atom number N_T. One therefore has

$$N = N_0 + N_T = N_0 + \underbrace{\sum_{i>0} \bar{n}_i(\mu, T)}_{\equiv N_T}. \tag{3.9}$$

The saturation of the particle number in the thermal cloud implies the existence of the Bose–Einstein condensate—a macroscopic phenomenon of bosons occupying the same single-particle state.

The average state occupation is a function of the chemical potential and temperature, two thermodynamical parameters important for modelling the dynamics of finite-temperature Bose gases. We will explore their calculation further in chapter 6.

One can define the critical temperature T_C of the Bose gas as the maximum temperature permitting Bose–Einstein condensation. For N atoms of mass m in a volume $\mathcal{V}$, the critical temperature is given by [PS16]

$$T_{\mathrm{C}} = \frac{h^2}{2\pi m k_{\mathrm{B}}} \left(\frac{N}{\zeta(3/2)\mathcal{V}} \right)^{\frac{2}{3}}, \tag{3.10}$$

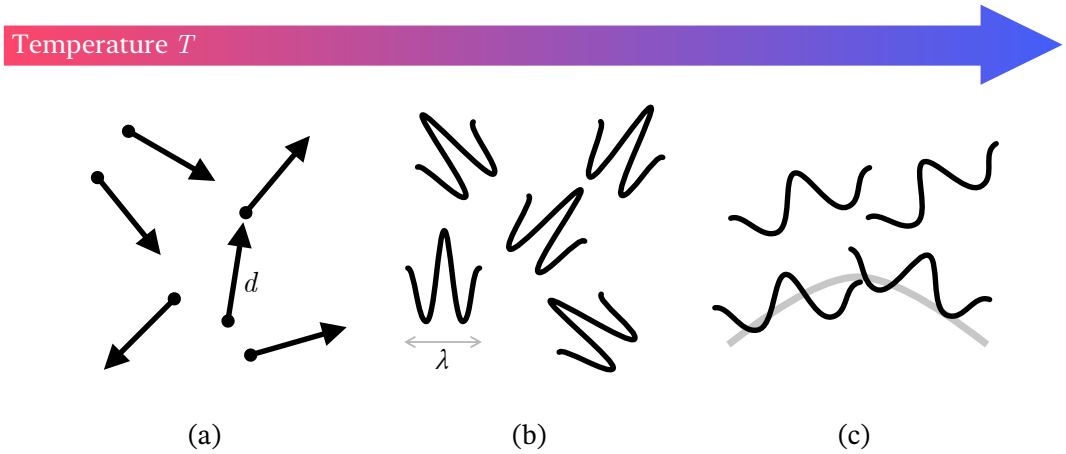


Figure 3.1: A schematic showing the effects of modelling a gas at temperatures $T \gg T_c$ in (a), $T > T_c$ in (b) and $T \sim T_c$ in (c). Figure 3.1 (c) shows the onset of Bose–Einstein condensation—the formation of a single matter wave (light grey). [Figure adapted from various authors.]

where h is the Planck constant, k_B is the Boltzmann constant and ζ is the Riemann zeta function, evaluating at $\zeta(3/2) \approx 2.612$. The $\zeta(3/2)$ factor comes from integrating the Bose–Einstein distribution n_{BE} over all energies.

Far above T_C, the behaviour of the gas obeys classical particle mechanics—there are no quantum effects as the inter-particle spacing d is sufficiently large. As the gas cools (but remains above T_C), the inter-particle distance is almost of the same order as the particle's de Broglie wavelength λ. As $T \sim T_C$, $d \sim \lambda$ which marks the onset of Bose–Einstein condensation. Figure 3.1 shows these three regimes.

Several theoretical schemes exist to describe the dynamics of Bose–Einstein condensates at various temperature regimes. There is not in principle a 'best' theoretical approach; indicative guidelines help us to choose a theory. We refer the reader to the Ph.D. tutorial on finite-temperature models of Bose gases by Proukakis and Jackson [PJ08] for a detailed discussion of common theoretical schemes and their validity.

3.3 Dynamics of Bose gases at zero temperature

A theoretical treatment of a dilute Bose gas may start with the Hamiltonian

$$\hat{H} = \int \mathrm{d}\mathbf{x}\hat{\Psi}^\dagger(t,\mathbf{x})\hat{h}_0(\mathbf{x})\hat{\Psi}(t,\mathbf{x}) + \frac{1}{2}\iint \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{x}'\hat{\Psi}^\dagger(t,\mathbf{x})\hat{\Psi}^\dagger(\mathbf{x}',t)U(\mathbf{x}-\mathbf{x}')\hat{\Psi}(\mathbf{x}',t)\hat{\Psi}(t,\mathbf{x}),$$

where

$$\hat{h}_0 = -\frac{\hbar^2\nabla^2}{2m} + V_{\text{ext.}}(t,\mathbf{x}) \tag{3.11}$$

is a single-particle operator in a (possibly time-dependent) external trapping potential $V_{\text{ext.}}$, $U(\mathbf{x}-\mathbf{x}')$ is a two-body interatomic potential, and the factor of $1/2$ in the second integral avoids double-counting particles. The Bose field operators

$$\hat{\Psi}(t,\mathbf{x}) = \sum_{k=1}^{\infty} \hat{a}_k(t)\phi_k(t,\mathbf{x}) \tag{3.12}$$

and

$$\hat{\Psi}^\dagger(t,\mathbf{x}) = \sum_{k=1}^{\infty} \hat{a}_k^\dagger(t)\phi_k^*(t,\mathbf{x}) \tag{3.13}$$

obey the commutation relations

$$[\hat{\Psi}(t,\mathbf{x}),\hat{\Psi}^\dagger(t,\mathbf{x}')] = \delta(\mathbf{x}-\mathbf{x}') \tag{3.14}$$

$$[\hat{\Psi}(t,\mathbf{x}),\hat{\Psi}(t,\mathbf{x}')] = [\hat{\Psi}^\dagger(t,\mathbf{x}),\hat{\Psi}(t,\mathbf{x}')] = 0, \tag{3.15}$$

and the creation and annihilation operators obey the commutation relations

$$[\hat{a}_\mu,\hat{a}_\nu^\dagger] = \delta_{\mu\nu} \tag{3.16}$$

$$[\hat{a}_\mu,\hat{a}_\nu] = [\hat{a}_\mu^\dagger,\hat{a}_\nu^\dagger] = 0. \tag{3.17}$$

We will now make some comments on the term $U(\mathbf{x}-\mathbf{x}')$. Despite the short-range interactions of alkali atoms being very strong, it is possible to consider an

ultracold gas of these atoms as weakly interacting [Bur96]. At very low temperatures, the de Broglie wavelength λ of the particles is much larger than the range of the interatomic potential; within good approximation, one can consider the interatomic interactions as two-body (contact) interactions characterised by the *s*-wave scattering length. The interactions are elastic, point-like and local, which can be described by a pseudopotential characterised by a Dirac delta function

$$U(\mathbf{x}-\mathbf{x}') \to U_{\mathrm{PS}} = g\delta(\mathbf{x}-\mathbf{x}), \tag{3.18}$$

where g is a parameter controlling the strength of interactions. The interaction strength is directly proportional to the *s*-wave scattering length

$$g = \frac{4\pi\hbar^2 a_s}{m}. \tag{3.19}$$

Using the pseudopotential in equation (3.18) in the original definition of the Hamiltonian in equation (3.3), one has

$$\hat{H} = \int \mathrm{d}\mathbf{x}\hat{\Psi}^\dagger(t,\mathbf{x})\hat{h}_0(\mathbf{x})\hat{\Psi}(t,\mathbf{x}) + \frac{g}{2}\int \mathrm{d}\mathbf{x}\hat{\Psi}^\dagger(t,\mathbf{x})\hat{\Psi}^\dagger(t,\mathbf{x})\hat{\Psi}(t,\mathbf{x})\hat{\Psi}(t,\mathbf{x}).$$

The field operators evolve according to the Heisenberg equation of motion.[1] For a general operator $\hat{A}_H$, the equation of motion is

$$i\hbar\frac{\partial\hat{A}_H}{\partial t} = [\hat{A}_H(t), \hat{H}]. \tag{3.20}$$

[1] In contrast to the Schrödinger picture, operators in the Heisenberg picture have time dependence, and state vectors are time-independent.

By expanding the commutator according to the Heisenberg equation of motion, one has

$$
\begin{aligned}
[\hat{\Psi}(t,\mathbf{x}),\hat{H}] = & \\
=&\hat{\Psi}(t,\mathbf{x})\Big(\int \mathrm{d}\mathbf{x}'\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{h}_0(\mathbf{x}')\hat{\Psi}(t,\mathbf{x}')\Big) \\
&-\Big(\int \mathrm{d}\mathbf{x}'\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{h}_0(\mathbf{x}')\hat{\Psi}(t,\mathbf{x}')\Big)\hat{\Psi}(t,\mathbf{x}) \\
&+\frac{g}{2}\Big[\hat{\Psi}(t,\mathbf{x})\Big(\int \mathrm{d}\mathbf{x}'\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{\Psi}(t,\mathbf{x}')\hat{\Psi}(t,\mathbf{x}')\Big) \\
&\qquad-\Big(\int \mathrm{d}\mathbf{x}'\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{\Psi}(t,\mathbf{x}')\hat{\Psi}(t,\mathbf{x}')\Big)\hat{\Psi}(t,\mathbf{x})\Big] \\
=&\int \mathrm{d}\mathbf{x}'\hat{\Psi}(t,\mathbf{x})\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{h}_0(\mathbf{x}')\hat{\Psi}(t,\mathbf{x}') \\
&-\int \mathrm{d}\mathbf{x}'\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{\Psi}(t,\mathbf{x})\hat{h}_0(\mathbf{x}')\hat{\Psi}(t,\mathbf{x}') \\
&+\frac{g}{2}\Big[\int \mathrm{d}\mathbf{x}'\hat{\Psi}(t,\mathbf{x})\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{\Psi}(t,\mathbf{x}')\hat{\Psi}(t,\mathbf{x}') \\
&\qquad-\int \mathrm{d}\mathbf{x}'\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{\Psi}(t,\mathbf{x})\hat{\Psi}(t,\mathbf{x}')\hat{\Psi}(t,\mathbf{x}')\Big] \\
=&\int \mathrm{d}\mathbf{x}'\big[\hat{\Psi}(t,\mathbf{x}),\hat{\Psi}^\dagger(t,\mathbf{x}')\big]\hat{h}_0(\mathbf{x}')\hat{\Psi}(t,\mathbf{x}') \\
&+\frac{g}{2}\int \mathrm{d}\mathbf{x}'\big[\hat{\Psi}(t,\mathbf{x}),\hat{\Psi}^\dagger(t,\mathbf{x}')\hat{\Psi}^\dagger(t,\mathbf{x}')\big]\hat{\Psi}(t,\mathbf{x}')\hat{\Psi}(t,\mathbf{x}').
\end{aligned}
\tag{3.21}
$$

Using the commutator relations in equation (3.15), one can determine the equation of motion for the Bose field, $\hat{\Psi}$, as

$$
i\hbar\frac{\partial\hat{\Psi}(t,\mathbf{x})}{\partial t} = \hat{h}_0\hat{\Psi}(t,\mathbf{x}) + g\hat{\Psi}^\dagger(t,\mathbf{x})\hat{\Psi}(t,\mathbf{x})\hat{\Psi}(t,\mathbf{x}). \tag{3.22}
$$

One can proceed further by decomposing the field operator $\hat{\Psi}(t,\mathbf{x})$ into two contributions: a field operator for the condensate atoms $\hat{\Phi}(t,\mathbf{x})$ and a field operator for the non-condensate atoms $\hat{\delta}(t,\mathbf{x})$ [Fet72]

$$
\hat{\Psi}(t,\mathbf{x}) = \hat{\Phi}(t,\mathbf{x}) + \hat{\delta}(t,\mathbf{x}). \tag{3.23}
$$

The non-condensed atoms may be thermally excited atoms, quantum fluctuations, or a mixture of the two. In the limit of a large number of condensed atoms, one can reduce the ensemble average of the Bose field operators to a classical field

$$\left\langle \hat{\Psi}(t,\mathbf{x}) \right\rangle = \Phi(t,\mathbf{x}). \tag{3.24}$$

Therefore,

$$\hat{\Psi}(t,\mathbf{x}) = \Phi(t,\mathbf{x}) + \hat{\delta}(t,\mathbf{x}), \tag{3.25}$$

which states that the Bose field operator is a function describing the mean of the quantum field and an operator describing the fluctuations about the mean. Inserting the field decomposition in equation (3.23) into the Heisenberg equation of motion (3.22) and taking the expected value of the field gives a non-linear Schrödinger equation describing the condensate and its (thermal or quantum) fluctuations

$$\begin{aligned} i\hbar\frac{\partial\langle\hat{\Psi}(t,\mathbf{x})\rangle}{\partial t} &= i\hbar\frac{\partial\Phi(t,\mathbf{x})}{\partial t} \\ &= \hat{H}_0(t,\mathbf{x})\Phi(t,\mathbf{x}) + g|\Phi(t,\mathbf{x})|^2\Phi(t,\mathbf{x}) + 2g\left\langle\hat{\delta}^\dagger(t,\mathbf{x})\hat{\delta}(t,\mathbf{x})\right\rangle\Phi(t,\mathbf{x}) \\ &\quad + g\langle\hat{\delta}(t,\mathbf{x})\hat{\delta}(t,\mathbf{x})\rangle\Phi^*(t,\mathbf{x}) + g\left\langle\hat{\delta}^\dagger(t,\mathbf{x})\hat{\delta}(t,\mathbf{x})\hat{\delta}(t,\mathbf{x})\right\rangle. \end{aligned} \tag{3.26}$$

When there are no fluctuations, i.e., $\hat{\delta} = 0$, equation (3.26) reduces to the Gross–Pitaevskii equation

$$i\hbar\frac{\partial\Phi(t,\mathbf{x})}{\partial t} = \underbrace{\left[-\frac{\hbar^2}{2m}\nabla^2 + V(t,\mathbf{x}) + g|\Phi(t,\mathbf{x})|^2\right]}_{\equiv\hat{H}_{\mathrm{GP}}}\Phi(t,\mathbf{x}). \tag{3.27}$$

The Gross–Pitaevskii equation is valid when the condensate is

- in the limit of large atom number
- weakly interacting and
- sufficiently cold (i.e., at or near zero temperature, typically valid when $T \lesssim T_C/2$).

Time-independent solutions of the Gross–Pitaevskii equation are permitted, which satisfy

$$\Phi(t, \mathbf{x}) = \phi(\mathbf{x})\exp\left(-\frac{i\mu t}{\hbar}\right). \tag{3.28}$$

Inserting equation (3.28) into equation (3.27) leads to the time-independent Gross–Pitaevskii equation

$$\mu\phi(\mathbf{x}) = -\frac{\hbar^2}{2m}\nabla^2\phi(\mathbf{x}) + V(\mathbf{x})\phi(\mathbf{x}) + g|\phi(\mathbf{x})|^2\phi(\mathbf{x}), \tag{3.29}$$

which is an eigenvalue equation where $\phi(\mathbf{x})$ are the stationary eigenstates of the system and μ, the chemical potential, is the eigenvalue.

In a homogenous condensate, the stationary solution is uniform and equation (3.29) reduces to

$$\mu\phi(\mathbf{x}) = g|\phi(\mathbf{x})|^2\phi(\mathbf{x}), \tag{3.30}$$

with solutions

$$\phi(\mathbf{x}) \equiv \phi_0 = \sqrt{\frac{\mu}{g}}, \tag{3.31}$$

Equation (3.31) is a helpful expression to check numerical implementations of the Gross–Pitaevskii equation and the (soon to be introduced) stochastic methodology—the quantity $n_0 = |\phi_0|^2$ gives an estimate of the peak atom density.

3.3.1 Trapping potentials

Figure 3.2 shows two common examples of trapping potential: the harmonic and toroidal trap.

Harmonic trap

The simplest confinement of atomic gases is in a harmonic trap

$$V(x, y, z) = \frac{1}{2}m(\omega_x^2 + \omega_y^2 + \omega_z^2)\rho(x, y, z)^2, \tag{3.32}$$

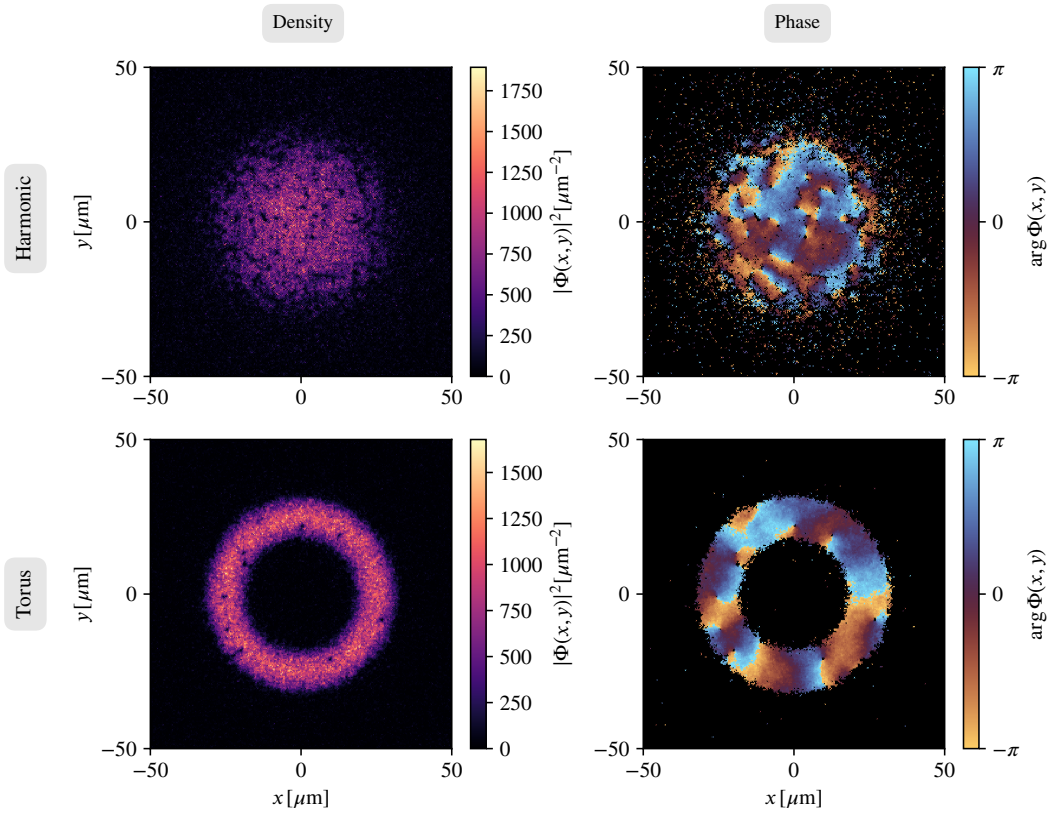


Figure 3.2: A harmonically trapped condensate (top) and a toroidally trapped condensate (bottom) at finite-temperature. The density (left) and phase (right) information are also shown. The phase is masked to remove the background density fluctuations. Regions of local phase coherence are visible in both condensates, with phase windings associated with vortices visible in both.

where $\rho^2 = x^2 + y^2 + z^2$ and ω_x, ω_y and ω_z are the frequencies of the trap in their respective dimensions. The harmonic trap has characteristic length scales

$$\ell_x = \sqrt{\frac{\hbar}{m\omega_x}}, \quad \ell_y = \sqrt{\frac{\hbar}{m\omega_y}} \quad \text{and} \quad \ell_z = \sqrt{\frac{\hbar}{m\omega_z}}, \tag{3.33}$$

which are referred to as the harmonic oscillator lengths.

Toroidal trap

The toroidal trap is

$$V(x,y,z) = V_0\left\{1 - \exp\left[-\frac{1}{\sigma^2}(\rho(x,y,z) - R)^2\right]\right\}, \tag{3.34}$$

where V_0 is the depth of the trap, ρ is the radial distance from the centre of the torus, and σ and R are the minor and major radii of the torus.

3.3.2 Hydrodynamical interpretation

The wave function may be written in its polar form (known in quantum hydrodynamics as the Madelung transform)

$$\Phi(t,\mathbf{x}) = |\Phi(t,\mathbf{x})|e^{i\theta(t,\mathbf{x})} = \sqrt{n(t,\mathbf{x})}e^{i\theta(t,\mathbf{x})}, \tag{3.35}$$

where $n(t,\mathbf{x})$ is the atomic number density and $\theta(t,\mathbf{x})$ is the phase of the condensate. The phase is related to the velocity distribution

$$\mathbf{v}(t,\mathbf{x}) = \frac{\hbar}{m}\nabla\theta(t,\mathbf{x}). \tag{3.36}$$

This implies that in this quantum-degenerate regime, the condensate velocity is irrotational

$$\nabla \times \mathbf{v} = \mathbf{0}. \tag{3.37}$$

Consider an arbitrary closed path $\mathbf{x} : [a,b] \to C$ through a toroidal geometry. Integrating the velocity in equation (3.36) over C gives the circulation

$$\Gamma = \oint_C \mathbf{v}(t,\mathbf{x}) \cdot \mathrm{d}\boldsymbol{\ell}. \tag{3.38}$$

Since the path is closed and the wave function is single-valued—but can vary by some phase factor—it is noted that the circulation is quantised

$$\Gamma = \frac{\hbar}{m} \oint_C \nabla\theta \cdot \mathrm{d}\boldsymbol{\ell} = \frac{\hbar}{m} 2\pi q, q \in \mathbb{Z}^{\pm}. \tag{3.39}$$

For a geometry where $\mathbf{x}(a) = \mathbf{x}(b)$, $\Gamma = 0$ and the geometry is referred to as *simply connected*. Geometries where $\Gamma \neq 0$ are referred to as multiply connected. Only a multiply connected geometry may accept non-zero quantisation of circulation—since we have introduced a phase singularity, and only a multiply connected geometry may permit the density vanishing at this singularity. Multiply connected geometries may be realised by the trapping potential (e.g., rings or lattices of rings) or through topological defects such as vortices.

Consider a general form of Stokes' theorem,

$$\nabla \times \mathbf{v} = 2\pi \frac{q\hbar}{m} \delta(\mathbf{x} - \mathbf{x}') \hat{\mathbf{e}}_z, \tag{3.40}$$

where $\mathbf{x}'$ is the position of a phase singularity. Equation (3.40) tells us that when a condensate rotates, it necessarily introduces phase singularities, which have been demonstrated experimentally [Mat+99; Mad+00; Abo+01]. Such systems are described as metastable; the energy cost is too high for a vortex to move from a torus's centre (density-depleted) region through the high-density atomic cloud. Vortices are otherwise unstable, and leave the condensate by moving towards low density regions.

3.3.3 Condensates of reduced dimensionality

In the present work, we mostly work with two-dimensional condensates (at zero or finite-temperature); we assume that there is sufficiently tight confinement in

one of the dimensions such that all dynamics are approximately suppressed in that dimension. Assuming tight confinement in the z-dimension requires that

$$\hbar\omega_z \gg (\mu, k_B T) \quad \text{and} \tag{3.41}$$

$$\hbar\omega_x, \hbar\omega_y \ll \hbar\omega_z. \tag{3.42}$$

The non-dynamical contribution to the wave function from the z-dimension is the ground state harmonic oscillator $\phi(z) = (\pi\ell_z)^{-1/4}\mathrm{e}^{-z^2/2\ell_z^2}$; the complete field is

$$\Phi(t, x, y, z) = \Phi_{2\mathrm{D}}(t, x, y)\phi(z). \tag{3.43}$$

The effective two-dimensional interaction strengths and chemical potentials are

$$g_{2\mathrm{D}} = \frac{g}{\sqrt{2\pi}\ell_z} \quad \text{and} \tag{3.44}$$

$$\mu_{2\mathrm{D}} = \mu - \frac{\hbar\omega_z}{2}. \tag{3.45}$$

We refer to the resulting condensate as being *quasi*-2D (i.e., the dynamics of the condensate are only in two dimensions, and the other dimension remains in the ground state for all time).

Similar arguments may be made for considering quasi-1D condensates. By considering dynamics only in the z-dimension and locking the x- and y-dimensions into their respective harmonic oscillator ground states, one can show that the effective interaction strength and chemical potentials are

$$g_{1\mathrm{D}} = \frac{g}{2\pi\ell_x\ell_y} \quad \text{and} \tag{3.46}$$

$$\mu_{1\mathrm{D}} = \mu - \frac{\hbar\omega_x}{2} - \frac{\hbar\omega_y}{2}. \tag{3.47}$$

3.4 Dynamics of Bose gases at finite-temperature

The Gross–Pitaevskii equation is successful at describing the dynamics of Bose–Einstein condensates at zero or near-zero temperatures (in regimes where the temperature of the thermal bath is approximately less than half of the critical temperature in equation (3.10)). In these temperature regimes we only need to consider the mean field (Gross–Pitaevskii equation).

Studying the dynamics of Bose gases at finite-temperatures is more numerically and theoretically involved. A Bose gas at finite-temperatures (where the usual Gross–Pitaevskii methodology is not appropriate) features a condensate component coupled statically or dynamically to a non-condensate component (below the critical temperature). The condensate and non-condensate components form a thermodynamical system with a constant exchange of atoms between the components. The nature of the interaction between the condensate and non-condensate modes is a *fluctuation-dissipation* theorem, a standard concept from statistical physics.

3.4.1 Brief comparison of theories

As the study of finite-temperature and non-equilibrium quantum gases has developed, many descriptions of their dynamics have emerged, often from very different theoretical approaches.

The *Zaremba–Nikuni–Griffin* (or *ZNG*) theory [ZNG99] provides a self-consistent treatment of the condensate and non-condensate modes in finite-temperature Bose gases. This approach dynamically couples a dissipative Gross–Pitaevskii equation, which describes the condensate mode, with a quantum Boltzmann equation, which describes the thermal bath. The self-consistency arises from the mutual influence between these components: the condensate's mean-field affects the thermal cloud's evolution, while the thermal cloud impacts the condensate through both mean-field effects and collisional processes. This coupled system of equations ensures that the dynamics of both components evolve in a mutually consistent manner.

The *stochastic Gross–Pitaevskii equation* is similar in principle to—and may be regarded as a generalisation of—the ZNG theory insofar that there is a coupling between the lower-lying modes (i.e., the condensate and all modes affected by its presence) and higher-lying modes. The key difference is that the evolution of these modes is now stochastic—thermal fluctuations are incorporated explicitly by a noise term (this may therefore be considered to be a dissipative Gross–Pitaevskii equation with additive noise). There are two theoretical formulations of this, developed independently by Stoof, Duine and Bijlsma [Sto97; Sto99; SB01; DS01] and Gardiner, Zoller and co-workers [GZ00; GZ97; GZ98; GAF02; GD03]. The former approaches the problem from quantum field theory whereas the latter approaches it from a quantum optics perspective. Nevertheless, the theories are effectively equivalent. In this thesis, we will focus exclusively on the methodology developed by Stoof, Duine and Bijlsma.

In the stochastic Gross–Pitaevskii methodologies, the thermal bath may be coupled statically or dynamically (via a quantum Boltzmann equation) to the lower-energy and condensate modes. In contrast, the ZNG theory requires dynamical coupling of the two components (again via a quantum Boltzmann equation). Throughout this thesis, we will consider only the stochastic Gross–Pitaevskii equation with a static thermal cloud.

The stochastic Gross–Pitaevskii equation obeys stochastic dynamics, which we discuss in appendix B. A partially Bose-condensed gas obeys a fluctuation-dissipation theorem. The fluctuation-dissipation theorem ensures no growth or evaporation of the condensate or the thermal cloud—the system relaxes to its correct equilibrium state. We will observe that the mathematics described in appendix B has similarities with the theoretical description of a finite-temperature Bose gas, which we will introduce now.

3.4.2 Stochastic Gross–Pitaevskii equation

Using the Keldysh non-equilibrium formalism [Kel64], Stoof derived a Fokker–Planck equation [Sto99] which can be reduced to a Langevin equation [SB01]

$$i\hbar\frac{\partial\Phi(t,\mathbf{x})}{\partial t} = \left[-\frac{\hbar^2}{2m}\nabla^2 + V(t,\mathbf{x}) - \mu - iR(t,\mathbf{x}) + g|\Phi(t,\mathbf{x})|^2\right]\Phi(t,\mathbf{x}) + \eta(t,\mathbf{x}), \tag{3.48}$$

where $\Phi(t,\mathbf{x})$ is the classical field describing the condensate and all modes affected by its presence. Compared to the Gross–Pitaevskii equation (3.27), this equation has two notable additions. The term $iR(t,\mathbf{x})$ describes the gain or loss of atoms due to collisions which transfer atoms between the condensate and thermal cloud—it is associated with dissipative effects. The term $\eta(t,\mathbf{x})$ is a dynamical noise term associated with the condensate fluctuations.

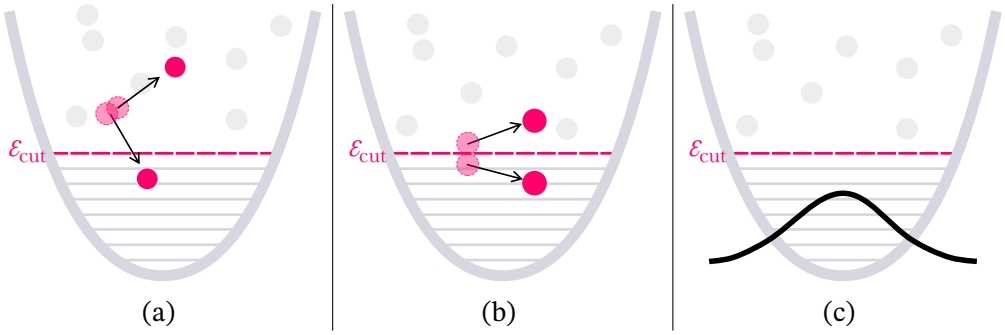


Figure 3.3: A schematic showing the stochastic model of the condensate growth. In (a), all atoms are purely thermal—there is no condensate. Non-condensate atoms may collide, however, and be scattered into the lower-lying modes, as shown in (b). Eventually, as more atoms are scattered and condense, the Bose–Einstein condensate forms as shown in (c).

In analogy to equation (B.7), the noise term is also assumed to obey the white noise approximation and has correlation functions

$$\langle\eta^*(t,\mathbf{x})\eta(t,\mathbf{x})\rangle = i\left(\frac{\hbar^2}{2}\right)\Sigma^K(t,\mathbf{x})\delta(\mathbf{x}-\mathbf{x}')\delta(t-t'), \tag{3.49}$$

where $\Sigma^K(t,\mathbf{x})$ is the Keldysh self-energy which arises from the scattering of atoms into or out of the thermal cloud. The Keldysh self-energy—a concept from quantum field theory—is the energy a particle has due to interactions between itself and the system [FW03]. The magnitude of the fluctuations is set by $\hbar\Sigma^K(t,\mathbf{x})$.

The fluctuation-dissipation theorem is described by

$$iR(t,\mathbf{x}) = -\frac{1}{2}\hbar\Sigma^K(t,\mathbf{x})\left[\frac{1}{1+2n_{\mathrm{BE}}(\hat{H}_{\mathrm{GP}})}\right], \tag{3.50}$$

where n_{BE} is the Bose–Einstein distribution.

Considering the first-order Taylor expansion of the Bose–Einstein distribution describing the thermal cloud, and in the limit of a large number of bosons, one finds that the distribution is actually—in excellent approximation—a Rayleigh-Jeans distribution [SB01]

$$\frac{1}{1+2n_{\mathrm{BE}}(\hat{H}_{\mathrm{GP}})} \approx \frac{\beta(\hat{H}_{\mathrm{GP}}-\mu)}{2}. \tag{3.51}$$

The fluctuation-dissipation theorem is therefore

$$iR(t,\mathbf{x}) \approx -\frac{\beta}{4}\hbar\Sigma^K(t,\mathbf{x})(\hat{H}_{\mathrm{GP}}-\mu). \tag{3.52}$$

It may be noted that the damping parameter γ from the original description of Langevin equations is

$$\gamma = i\frac{\beta\hbar\Sigma^K(t,\mathbf{x})}{4}. \tag{3.53}$$

For near-equilibrium harmonically trapped condensates,

$$\gamma \approx \text{few} \times \frac{4ma_s k_B T}{\pi\hbar^3} \ll 1, \tag{3.54}$$

where the pre-factor of a few is of order unity. The spatial dependence in γ is typically ignored. Moreover, whilst technically only derived for harmonically trapped condensates, the relation matches experimentally-observed growth of condensates for most uniform traps. The damping parameter may be treated as a fitting parameter

to mimic experimental growth [Wei+08]. The approximation in equation (3.54) leads to the stochastic Gross–Pitaevskii equation

$$i\hbar\frac{\partial\Phi(t,\mathbf{x})}{\partial t} = (1-i\gamma)\left[-\frac{\hbar^2}{2m}\nabla^2 + V(t,\mathbf{x}) + g|\Phi(t,\mathbf{x})|^2 - \mu\right]\Phi(t,\mathbf{x}) + \eta(t,\mathbf{x}), \quad (3.55)$$

with Gaussian noise ensemble correlations

$$\langle\eta^*(t,\mathbf{x})\eta(t,\mathbf{x})\rangle = 2\gamma\hbar k_B T\delta(\mathbf{x}-\mathbf{x}')\delta(t-t'). \quad (3.56)$$

3.4.3 Time of flight measurement of the temperature

The chemical potential, μ, and temperature, T, are inputs to the stochastic Gross–Pitaevskii equation. Experimentalists typically only worry about ballpark values for these values (if they are interested at all). Sometimes the temperature is measured using more sophisticated fitting techniques, such as during a time of flight procedure.

In an experiment, any optical or magnetic traps are switched off, and the condensate is allowed to fall in free space. Assume that the atomic cloud obeys a Gaussian density distribution. By measuring first the $1/e^2$ width of the trapped atomic cloud, σ_0, then measuring the $1/e^2$ width of the falling cloud, σ, at some time t, one can find a linear correlation between σ^2 and t^2 which allows one to obtain the temperature,

$$\sigma^2 = \sigma_0^2 + \frac{k_B T}{m}t^2. \quad (3.57)$$

This is fundamentally still an approximation. Performing time of flight measurements also destroys the sample. Measuring temperature experimentally is difficult, and measuring the chemical potential is even harder (there is no single equation to determine it, as it changes for every geometry).

3.4.4 Condensate growth and equilibrium solutions

The equilibrium properties of finite-temperature Bose gases are mostly insensitive to the dynamical means of reaching equilibrium, and the choice of theory (including whether or not to include a projector) should not be a major concern when finding equilibrium states [PDG13]. After equilibrium, subsequent dynamics may have greater sensitivity to the chosen method.

Figures 3.4–3.5 show the evolution of the density and phase, respectively, of a harmonically trapped, quasi-2D condensate evolved according to the stochastic Gross–Pitaevskii equation. Figures 3.6–3.7 show the evolution of the density and phase, respectively, of a toroidally trapped, quasi-2D condensate evolved according to the stochastic Gross–Pitaevskii equation. For details on the numerical schemes to implement the stochastic Gross–Pitaevskii equation—like in these figures—see § 3.5. One can observe the emergence of topological defects, such as vortices in the density and phase profiles. Over time, the condensate develops local phase coherence as vortices annihilate or leave the condensate by minimising the system's angular momentum.

3.4.5 Role of the projection operator

In order for the approximation in equation (3.51) to hold (and, in turn, for the stochastic Gross–Pitaevskii equation to be valid), one requires that the highest-energy coherent mode is macroscopically occupied, i.e., that $n_{\mathrm{BE}} \sim 1$. We will define $\varepsilon_{\mathrm{cut}}$ as the energy of this highest coherent mode. Using the definition of the Bose–Einstein distribution in equation (3.8), it follows that

$$\varepsilon_{\mathrm{cut}} \approx k_B T \ln 2 + \mu. \tag{3.58}$$

All coherent modes below this cut-off energy, i.e., $\mathbf{C} = \{n : \varepsilon_n \leq \varepsilon_{\mathrm{cut}}\}$ are associated with the classical field Φ. All incoherent modes above this cut-off energy, i.e., $\mathbf{I} = \{n : \varepsilon_n \geq \varepsilon_{\mathrm{cut}}\}$ are associated with the thermal bath; constant equilibrium between

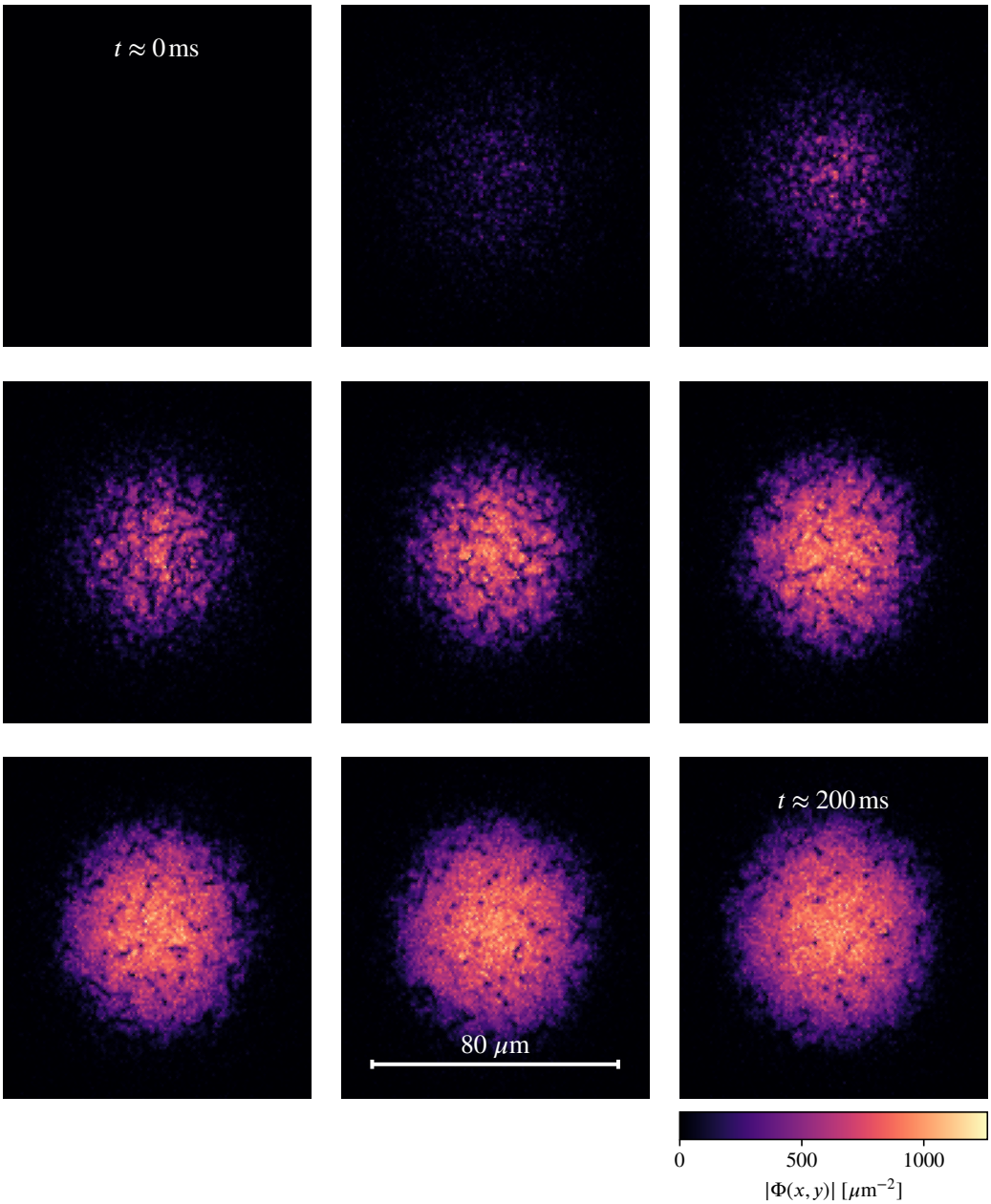


Figure 3.4: The growth of a harmonically trapped quasi-2D condensate from thermal noise to thermal equilibrium. The condensate has a chemical potential $\mu = 30$ nK and temperature $T = 30$ nK. The atomic densities share a common scale (earlier simulations are difficult to see since the atomic density is relatively low). These snapshots of the atomic density profile are not uniformly sampled—they are chosen to show important moments of the growth of the condensate.

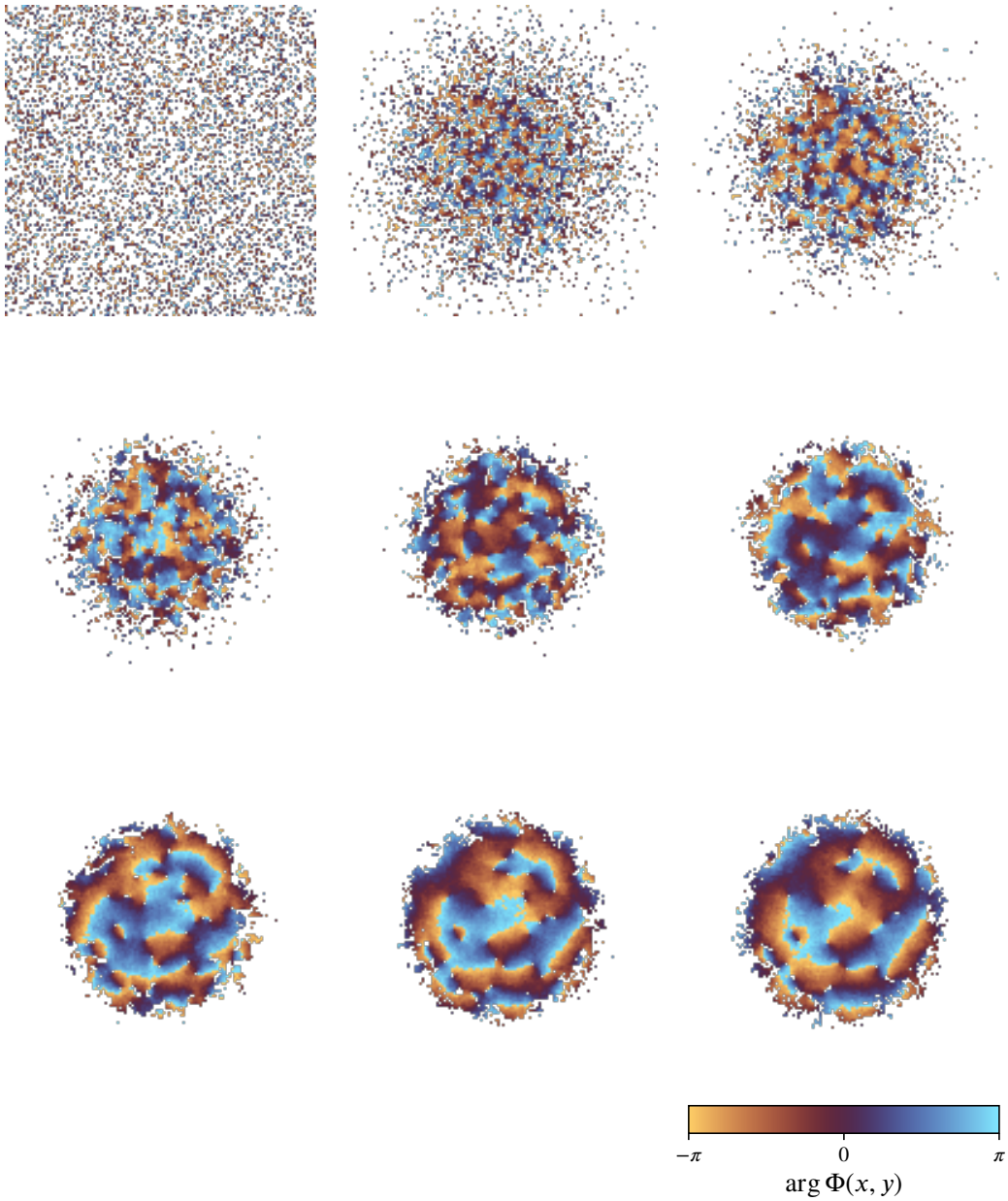


Figure 3.5: The evolution of the phase of a harmonically trapped quasi-2D condensate from thermal noise to thermal equilibrium. These phases correspond to the selected density profiles in Figure 3.4. The condensate has a chemical potential $\mu = 30$ nK and temperature $T = 30$ nK. Areas of local phase coherence begin to emerge as the condensate thermalises. Vortices can be observed in the phase profiles; there are areas of density depletion (the small white points in the interior of the condensate).

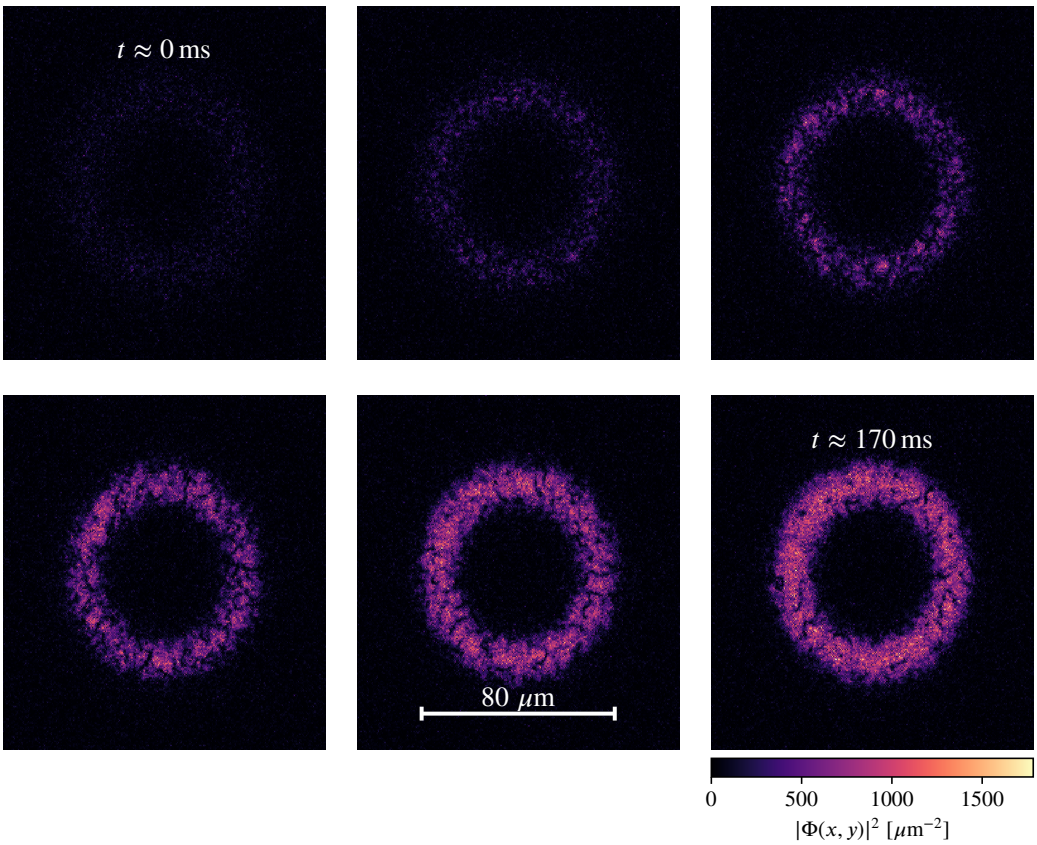


Figure 3.6: The growth of a toroidally trapped quasi-2D condensate from thermal noise to thermal equilibrium. The condensate has a chemical potential $\mu = 30$ nK and temperature $T = 30$ nK. The atomic densities share a common scale (earlier simulations are difficult to see since the atomic density is relatively low). These snapshots of the atomic density profile are not uniformly sampled—they are chosen to show important moments of the growth of the condensate.

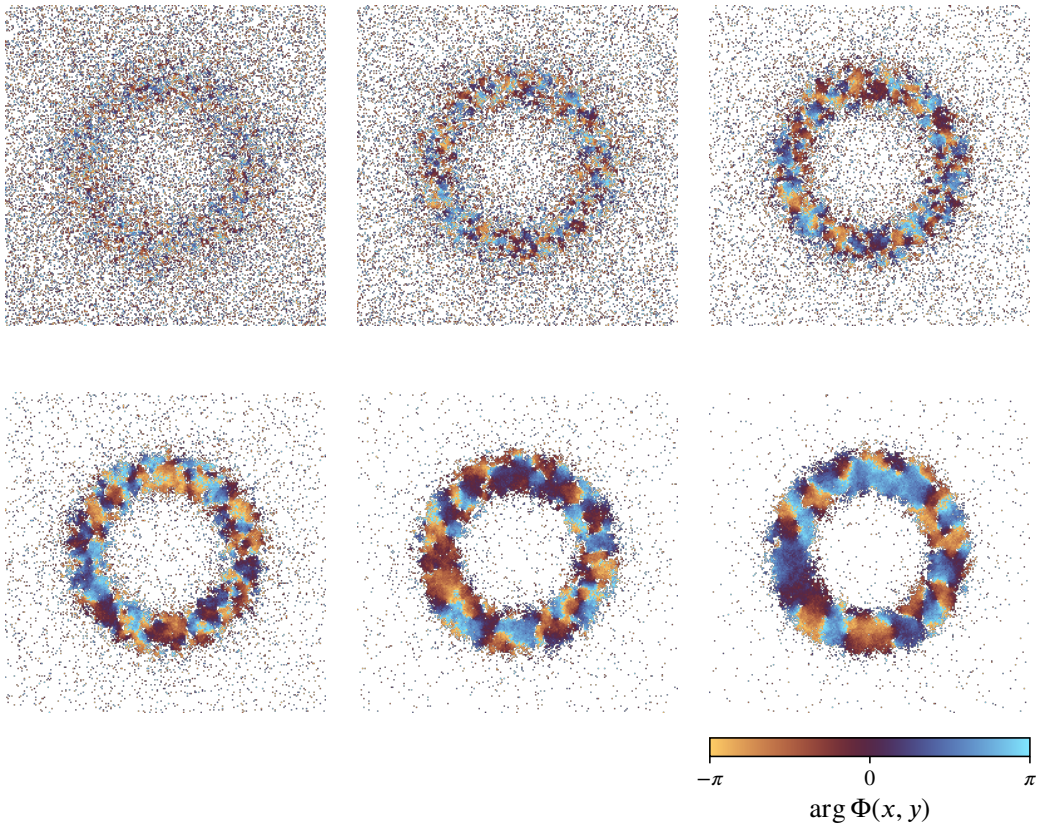


Figure 3.7: The evolution of the phase of a toroidally trapped quasi-2D condensate from thermal noise to thermal equilibrium. These phases correspond to the selected density profiles in Figure 3.4. The condensate has a chemical potential $\mu = 30$ nK and temperature $T = 30$ nK. Areas of local phase coherence begin to emerge as the condensate thermalises. Vortices can be observed in the phase profiles; there are areas of density depletion (the small white points in the interior of the condensate).

the coherent and incoherent modes is assumed. To enforce this energy cut-off, a projection operator $\hat{\mathcal{P}}$ is introduced such that

$$\hat{\mathcal{P}}\{\Phi(t,\mathbf{x})\} = \sum_{n\in\mathbf{C}} \zeta_n^*(\mathbf{x}) \int \mathrm{d}\mathbf{x}' \zeta_n^*(\mathbf{x}')\Phi(t,\mathbf{x}'), \tag{3.59}$$

where $\zeta_n(\mathbf{x})$ is the n-th single particle eigenstate of the classical field Φ.

Energy-cut off from numerical discretisation

Section 3.5 introduces a complete treatment of the numerical solution of the stochastic Gross–Pitaevskii equation. There is a critical implementation detail about the high-energy modes which exists at the intersection of the physics presented here and the numerical methods introduced in appendix C.

We discretise our spatial domains into grids of length L_x, L_y and L_z (there is also a discretisation of the temporal domain, which is unimportant for this discussion). Assuming that there are N_x, N_y and N_z gridpoints in each dimension, it follows that

$$\Delta x = \frac{N_x}{L_x}, \quad \Delta y = \frac{N_y}{L_y} \quad \text{and} \quad \Delta z = \frac{N_z}{L_z}. \tag{3.60}$$

Spatial discretisation inherently introduces a limit on the highest wave vector and associated energy that our simulations can represent, and a limit to the smallest wavelength that can be represented on the grid. In the x-dimension, for example,

$$\lambda_{\text{min}} = 2\Delta x, \tag{3.61}$$

which is associated with the maximum wave vector that can be represented on the grid

$$k_{\text{max}} = \frac{2\pi}{\lambda_{\text{min}}} = \frac{2\pi}{2\Delta x} = \frac{\pi}{\Delta x} \tag{3.62}$$

$$\implies \Delta x \leq \frac{\pi}{k_{\text{max}}}. \tag{3.63}$$

Equation (3.63) naturally leads to a maximum momentum $p_{\max} = \hbar k_{\max}$ that the grid can represent. We refer to this as an implicit projection. Without careful choice of spatial discretisation, high-frequency components will be indistinguishable from lower-frequency components—this is *aliasing*. Physically, the introduction of aliasing to simulations of the stochastic methodologies leads to unphysical collisions between non-momentum conserving modes. Arguments by Blakie, *et al.* [Bla+08] suggest that for the full numerical implementation of the cubic nonlinearity, one requires a grid sizing *twice* as stringent as the requirement in equation (3.63), i.e., that

$$\Delta x \leq \frac{\pi}{2k_{\max}}. \tag{3.64}$$

As noted before, the means of reaching equilibrium may be unimportant. An explicit projection of high-energy modes may only be necessary for studying subsequent dynamics after thermal equilibrium has been reached; implicit projection using grid methods may be sufficient for reaching thermal equilibrium (which this thesis only concerns).

Numerical implementation of the projector

The cut-off energy in equation (3.58) maps to a circle (2D) or sphere (3D) of radius

$$\hbar k_{\text{cut}} = \sqrt{2m\varepsilon_{\text{cut}}} \tag{3.65}$$

in momentum space. In Fourier space, modes with momentum less than this cut-off are retained by the projection operator (via a simple zero-or-one map).

3.5 Numerical implementation of theoretical schemes

An exciting moment has arrived! We will now discuss the foundations of the numerical implementation of the Gross–Pitaevskii equation and its stochastic counterpart.

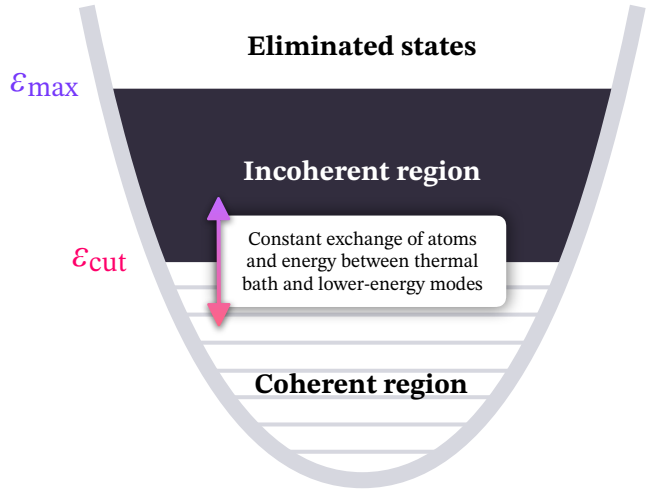


Figure 3.8: A schematic showing three regions in a harmonic trap: the coherent, low-energy/condensate modes; the incoherent, high-energy modes (which are cut out of simulations implicitly through a numerical scheme or explicitly through a projector); and those eliminated states which are beyond the *s*-wave approximation (which ultimately amounts to ensuring that the choice of the interaction pseudopotential remains valid).

When simulating the stochastic Gross–Pitaevskii equation, we used the Rust programming language [MK14]. Developers love Rust—it has been the most admired language on the Stack Overflow annual developer survey for eight consecutive years (as of 2023) [Sta24].[2] Rust is a fast and reliable language with many desirable features of a modern programming language: concurrency, memory safety, a dedicated package manager, excellent error messaging, and zero-cost abstractions. Our code is available freely [Gri24d].

3.5.1 Finding the ground state of a zero-temperature Bose gas

We first wish to find the dynamics of a zero- or near-zero temperature Bose–Einstein condensate, which obeys the Gross–Pitaevskii equation (3.27). The procedure is to obtain the ground state solution and evolve this over time with the Gross–Pitaevskii equation.

[2]This is in contrast to MATLAB, which is the least admired language. Less than 20% of developers who used MATLAB want to use it again in 2024, whereas 80% of developers who used Rust want to use it again in 2024.

A typical method to find the ground state is the *imaginary time* method. It is a computationally efficient method, but it is not always guaranteed to converge to the ground state. Consider the expansion of the field $\Phi(t, \mathbf{x})$ into a finite, complete, and orthonormal basis $\{\xi_n(\mathbf{x})\}$ weighted by time-dependent coefficients $\{a_n(t)\}$ and associated time-dependent energies $\{E_n(t)\}$

$$\Phi(t, \mathbf{x}) = \sum_{n=0}^{\infty} a_n(t)\xi_n(\mathbf{x})\exp\left(-\frac{iE_n t}{\hbar}\right). \tag{3.66}$$

The imaginary time method uses the Wick transform $t \to -i\tau$, so the field becomes

$$\Phi(-i\tau, \mathbf{x}) = \sum_{n=0}^{\infty} a_n(-i\tau)\xi_n(\mathbf{x})\exp\left(-\frac{E_n \tau}{\hbar}\right), \tag{3.67}$$

where the energies are monotonically increasing as n increases. Let E_0 be the system's ground state (lowest) energy. Since $E_0 < E_n \forall n > 0$, the basis states decay more rapidly as they become more energetic than the ground state. After sufficient simulation time, the solution is *approximately* the ground state solution, and the subsequent dynamics can then be modelled using the Gross–Pitaevskii equation.

3.5.2 Dimensionless form of the stochastic Gross–Pitaevskii equation

In order to avoid issues when computing very small terms (such as $\hbar^2$), we work in a system of units where all quantities are dimensionless. In particular, we introduce the following scaling quantities:

$$\text{length} \quad \ell_x = \sqrt{\frac{\hbar}{m\omega_x}}, \tag{3.68}$$

$$\text{time} \quad \tau = \frac{1}{\omega_x} \quad \text{and} \tag{3.69}$$

$$\text{temperature (energies)} \quad T = \frac{k_B}{\hbar\omega_x}. \tag{3.70}$$

The scaling quantities in equations (3.68–3.70) lead to the dimensionless quantities in table 3.1.

Quantity	Scaled Variable
Condensate field	$\tilde{\Phi}(t,\mathbf{x}) = \ell_x^{3/2}\Phi(t,\mathbf{x})$
Temperature	$\tilde{T} = \dfrac{k_B}{\hbar\omega_x}T$
Position	$\tilde{\mathbf{x}} = \dfrac{\mathbf{x}}{\ell_x}$
Time	$\tilde{t} = \dfrac{t}{\tau}$
Laplacian	$\tilde{\nabla}^2 = \nabla^2\ell_x^2$
Chemical potential	$\tilde{\mu} = \dfrac{\mu}{\hbar\omega_x}$
Interaction strength	$\tilde{g} = \dfrac{g}{\hbar\omega_x\ell_x^3}$

Table 3.1: Dimensionless quantities (denoted by a tilde) scaled by equations (3.68–3.70).

The scaled variables lead to the dimensionless form of the stochastic Gross–Pitaevskii equation (where we will henceforth omit the tildes)

$$i\frac{\partial\Phi(t,\mathbf{x})}{\partial t} = (1-i\gamma)\Big[-\frac{1}{2}\nabla^2 + V(t,\mathbf{x}) + g|\Phi(t,\mathbf{x})|^2 - \mu\Big]\Phi(t,\mathbf{x}) + \eta(t,\mathbf{x}), \qquad (3.71)$$

with Gaussian noise ensemble correlations

$$\langle\eta^*(t,\mathbf{x})\eta(t,\mathbf{x})\rangle = 2\gamma T\delta(\mathbf{x}-\mathbf{x}')\delta(t-t'). \qquad (3.72)$$

3.5.3 Implementing numerical noise

We have formulated the stochastic Gross–Pitaevskii equation (3.55) as a Langevin equation, allowing us to use the framework of stochastic dynamics as introduced in appendix B. In complete analogy with equation (B.5), one may solve the stochastic Gross–Pitaevskii equation as a first-order (in time), inhomogeneous differential equation with the integrating factor

$$M(t) = \mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)t} \equiv \mathrm{e}^{\frac{i}{\hbar}(1-i\gamma)(\hat{H}_{\mathrm{GP}}-\mu)t}, \tag{3.73}$$

where we have defined the constant $\Gamma \equiv (1 - i\gamma)$. Dividing the stochastic Gross–Pitaevskii equation (3.55) by $i\hbar$, rearranging to isolate the inhomogeneity on the right-hand side, and applying the integrating factor in equation (3.73) gives

$$M(t)\frac{\partial\Phi(t,\mathbf{x})}{\partial t} + M(t)\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}} - \mu)\Phi(t,\mathbf{x}) = -M(t)\frac{i}{\hbar}\eta(t,\mathbf{x}) \tag{3.74a}$$

$$\implies \mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)t}\left[\frac{\partial\Phi(t,\mathbf{x})}{\partial t} + \frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}} - \mu)\Phi(t,\mathbf{x})\right] = -\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)t}\frac{i}{\hbar}\eta(t,\mathbf{x}) \tag{3.74b}$$

$$\implies \frac{\partial}{\partial t}\left[\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)t}\Phi(t,\mathbf{x})\right] = -\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)t}\frac{i}{\hbar}\eta(t,\mathbf{x}). \tag{3.74c}$$

Let us describe the evolution of the condensate wave function from t to $t + \Delta t$. Integration over these limits gives

$$\int_t^{t+\Delta t} \mathrm{d}\tau\, \frac{\partial}{\partial\tau}\left[\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)\tau}\Phi(\tau,\mathbf{x})\right] = -\frac{i}{\hbar}\int_t^{t+\Delta t} \mathrm{d}\tau\, \mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)\tau}\eta(\tau,\mathbf{x}). \tag{3.75}$$

The left-hand side evaluates to give

$$\int_t^{t+\Delta t} \mathrm{d}\tau\, \frac{\partial}{\partial\tau}\left[\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)\tau}\Phi(\tau,\mathbf{x})\right] = \mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)(t+\Delta t)}\Phi(t+\Delta t,\mathbf{x}) - \mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)t}\Phi(t,\mathbf{x}) \tag{3.76}$$

Taking out the common factor of $\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)t}$ and dividing both sides of equation (3.75) by this common factor gives

$$\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)\Delta t}\Phi(t+\Delta t,\mathbf{x})-\Phi(t,\mathbf{x})=-\frac{i}{\hbar}\xi(t,\mathbf{x}), \tag{3.77}$$

where we have introduced a new noisy field

$$\xi(t,\mathbf{x})=\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)t}\int_{t}^{t+\Delta t}\mathrm{d}\tau\,\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)\tau}\eta(\tau,\mathbf{x}) \tag{3.78}$$

with its associated correlation function, which we will determine later. Therefore, given the condensate wave function at some time t, the wave function at the next time step $t+\Delta t$ is given by

$$\Phi(t+\Delta t,\mathbf{x})=\mathrm{e}^{-\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)\Delta t}\left[\Phi(t,\mathbf{x})-\frac{i}{\hbar}\xi(t,\mathbf{x})\right]. \tag{3.79}$$

Let us move from the continuous variable t to discrete time steps $t_m=m\Delta t$ and $t_n=t_m+\Delta t=\Delta t(m+1)$. We are only interested in a first-order approximation for the behaviour of the noise over one time step given by the interval $[t_m,t_n]$. Assuming that the time step is small compared to all other dynamical time scales, the non-linearity in $\hat{H}_{\mathrm{GP}}$ may be treated as a constant potential energy within this interval. Similarly, any time-dependence in the potential energy $V(t,\mathbf{x})$ becomes insignificant if it varies slowly over this interval. Given this, all operators in $\hat{H}_{\mathrm{GP}}$ are all approximately equal and commutative.

Let us construct the correlation function $\langle\xi_m^*(\mathbf{x})\xi_n(\mathbf{x})\rangle$, where we have removed the dependency upon the continuous temporal variable and labelled each ξ with its respective time step. Using equation (3.78), the correlation function is

$$\begin{aligned}\langle\xi_m^*(\mathbf{x})\xi_n(\mathbf{x}')\rangle=\Bigg\langle&\left(\mathrm{e}^{-\frac{i}{\hbar}\Gamma^*(\hat{H}_{\mathrm{GP}}-\mu)t_m}\int_{t_m}^{t_m+\Delta t}\mathrm{d}\tau'\,\mathrm{e}^{-\frac{i}{\hbar}\Gamma^*(\hat{H}_{\mathrm{GP}}-\mu)\tau'}\eta^*(\tau',\mathbf{x})\right)\\&\times\left(\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)t_n}\int_{t_n}^{t_n+\Delta t}\mathrm{d}\tau\,\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)\tau}\eta(\tau,\mathbf{x}')\right)\Bigg\rangle.\end{aligned} \tag{3.80}$$

We will assume that the noise is uncorrelated at different times using the same white noise assumptions as for η. We are therefore interested only in the autocorrelation function when $m = n$ (otherwise, the expectation value will be zero). It follows that

$$\begin{aligned}\langle \xi_m^*(\mathbf{x})\xi_m(\mathbf{x}')\rangle = \mathrm{e}^{-\frac{i}{\hbar}\Gamma^*(\hat{H}_{\mathrm{GP}}-\mu)t_m}\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)t_n}\times \\ \times\int_{t_m}^{t_m+\Delta t}\mathrm{d}\tau'\int_{t_n}^{t_n+\Delta t}\mathrm{d}\tau\,\mathrm{e}^{-\frac{i}{\hbar}\Gamma^*(\hat{H}_{\mathrm{GP}}-\mu)\tau'}\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)\tau}\langle\eta_m^*(\mathbf{x})\eta_m(\mathbf{x}')\rangle.\end{aligned} \tag{3.81}$$

Using the original definition of the noise η in equation (3.56), one has

$$\begin{aligned}\langle \xi_m^*(\mathbf{x})\xi_m(\mathbf{x}')\rangle = 2\gamma k_B T\mathrm{e}^{-\frac{i}{\hbar}\Gamma^*(\hat{H}_{\mathrm{GP}}-\mu)t_m}\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)t_m}\times \\ \times\int_{t_m}^{t_m+\Delta t}\mathrm{d}\tau'\int_{t_m}^{t_m+\Delta t}\mathrm{d}\tau\,\mathrm{e}^{-\frac{i}{\hbar}\Gamma^*(\hat{H}_{\mathrm{GP}}-\mu)\tau'}\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)\tau}\delta(\mathbf{x}-\mathbf{x}')\delta(\tau-\tau').\end{aligned} \tag{3.82}$$

The Dirac delta function $\delta(\tau-\tau')$ collapses the integral into a single temporal integral

$$\langle \xi_m^*(\mathbf{x})\xi_m(\mathbf{x}')\rangle = 2\gamma k_B T\int_{t_m}^{t_m+\Delta t}\mathrm{d}\tau\,\mathrm{e}^{-\frac{i}{\hbar}\Gamma^*(\hat{H}_{\mathrm{GP}}-\mu)(\tau-t_m)}\mathrm{e}^{\frac{i}{\hbar}\Gamma(\hat{H}_{\mathrm{GP}}-\mu)(\tau-t_m)}\delta(\mathbf{x}-\mathbf{x}'), \tag{3.83}$$

where we have additionally combined all the exponential terms. Within the time interval Δt, we note that $\mathrm{e}^{\hat{X}}\mathrm{e}^{\hat{Y}} \approx \mathrm{e}^{\hat{X}+\hat{Y}}$ (which is true only to first-order). Using the definition of $\Gamma = (1-i\gamma)$ and then simplifying the exponential terms in the integrand, one can show that their product is approximately equal to unity. Thus,

$$\begin{aligned}\langle \xi_m^*(\mathbf{x})\xi_m(\mathbf{x}')\rangle &\approx 2\gamma k_B T\int_{t_m}^{t_m+\Delta t}\mathrm{d}\tau\,\delta(\mathbf{x}-\mathbf{x}') \\ &= 2\gamma k_B T\delta(\mathbf{x}-\mathbf{x}')\int_{t_m}^{t_m+\Delta t}\mathrm{d}\tau \\ &= 2\gamma k_B T\delta(\mathbf{x}-\mathbf{x}')\Delta t.\end{aligned} \tag{3.84}$$

In a discrete, three-dimensional grid, the Dirac delta function

$$\delta(\mathbf{x}-\mathbf{x}') \equiv \delta(x-x')\delta(y-y')\delta(z-z') \to \frac{1}{\Delta x\Delta y\Delta z}, \tag{3.85}$$

where Δx, Δy and Δz are the spatial step-sizes. Thus, to a first approximation, the noise has correlation functions

$$\langle \xi_m^*(\mathbf{x}) \xi_m(\mathbf{x}') \rangle \approx 2\gamma k_B T \frac{\Delta t}{\Delta x \Delta y \Delta z}. \tag{3.86}$$

The magnitude of the discretised noise is the square root of this correlation function, i.e.,

$$\sigma = \sqrt{2\gamma k_B T \frac{\Delta t}{\Delta x \Delta y \Delta z}}. \tag{3.87}$$

We model the noise as a Wiener process, which is a continuous-time stochastic process that represents the integral of a Gaussian white noise process (discussed in appendix B).

The initial condition is complex, so we additionally sample an intermediate variable

$$\Theta(t, \mathbf{x}) = \mathrm{e}^{2i\pi\theta}, \quad \theta \sim \text{Uniform}(0, 1), \tag{3.88}$$

which is associated with the phase of the noise. The complex noise is, therefore, the product

$$\eta(t, \mathbf{x}) = \sigma W(t, \mathbf{x}) \times \Theta(t, \mathbf{x}), \tag{3.89}$$

which is sampled at every time step as the condensate grows (and beyond). In the SGPE or SPGPE, the complex noise seeds the initial condition $\Phi(t = 0, \mathbf{x}) = \eta(t = 0, \mathbf{x})$. The noise is dynamical, and it is updated at every time step, i.e., $\eta(t_m, \mathbf{x}) \neq \eta(t_m + \Delta t, \mathbf{x})$.

Algorithm 2 demonstrates the numerical implementation of the noise processes.

Figure 3.9 shows the average density and phase profiles for a harmonically and toroidally trapped Bose–Einstein condensate at a temperature $T = 80\,\mathrm{nK}$ using an ensemble of 100 different noise realisations. Topological defects such as vortices are smoothed out in the averaging process. Observable quantities such as the average atom number can be extracted from these averages.

Algorithm 2 Generate two-dimensional (complex-valued) noise for the stochastic Gross–Pitaevskii equation

```
1: procedure GenerateWienerNoise(N_x, N_y)
2:     for i ← 1 to N_x do
3:         for j ← 1 to N_y do
4:             W_ij ← SampleFromStandardNormal()
5:         end for
6:     end for
7:     return W
8: end procedure
9: procedure GeneratePhaseNoise(N_x, N_y)
10:    for i ← 1 to N_x do
11:        for j ← 1 to N_y do
12:            θ_ij ← SampleFromUniform(0, 1)
13:        end for
14:    end for
15:    return θ
16: end procedure
17: procedure CalculateNoise(σ, W, θ)
18:    for i ← 1 to N_x do
19:        for j ← 1 to N_y do
20:            η_ij ← σ W_ij exp(2πiθ_ij)
21:        end for
22:    end for
23:    return η
24: end procedure
25: procedure Main
26:    N_x, N_y ← GridDimensions
27:    σ ← NoiseMagnitude
28:    W ← GenerateWienerNoise(N_x, N_y)
29:    θ ← GeneratePhaseNoise(N_x, N_y)
30:    η ← CalculateNoise(σ, W, θ)
31:    return η
32: end procedure
```

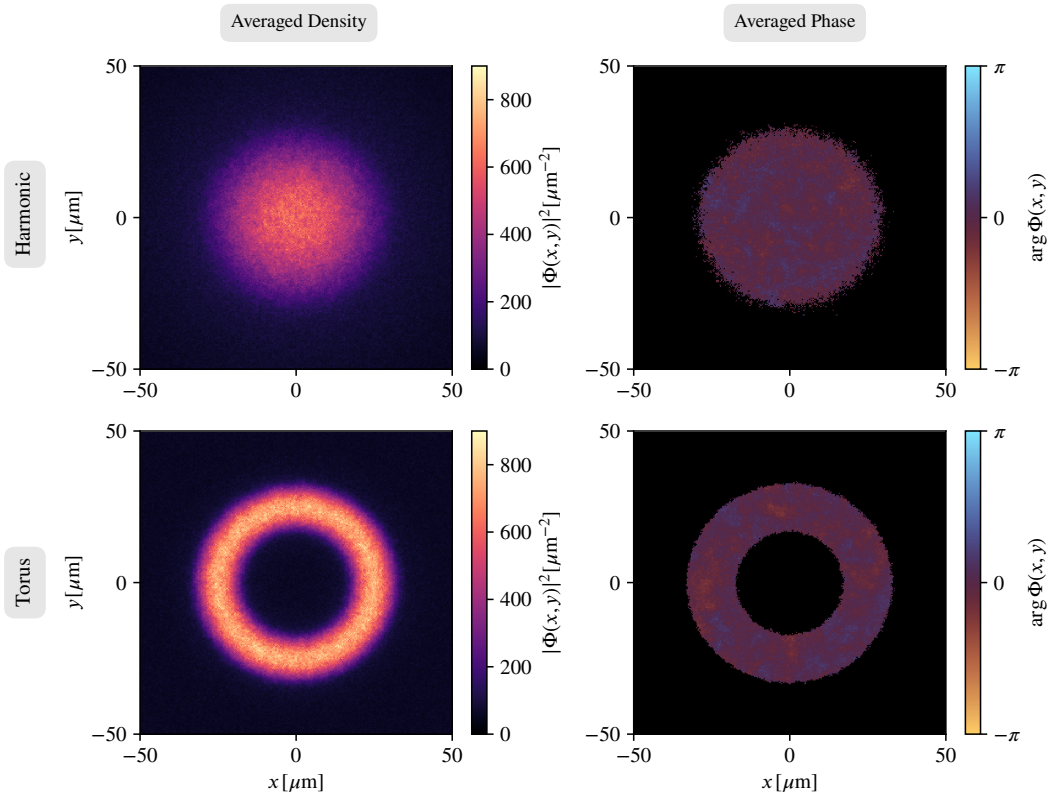


Figure 3.9: A harmonically trapped condensate (top) and a toroidally trapped condensate (bottom) at finite-temperatures. The chemical potential was 24 nK and the temperature was 80 nK. The density and phase profiles are an ensemble average over 100 different realisations of the noise.

We discuss numerical integration and differentiation in appendix C. We use the fourth-order Runge–Kutta method to evolve the SGPE in time. When probing the evolution of long dynamical routines, it is usually advantageous to use an adaptive step-size routine to speed up the numerical integration.[3] The noise is added after every step in the aforementioned integration routines $\Phi_{n+1} = \cdots + \eta_{n+1}$.

[3]It must be noted that adaptive step-size algorithms are typically not used for stochastic differential equations. However, as the SGPE and SPGPE are approximated as a Langevin equation with weak, additive stochastic noise, it is acceptable in this instance [MT04].

3.6 Summary

In this chapter, we have explored the implementation of zero temperature and finite-temperature schemes describing Bose–Einstein condensation. We explored the typical thermodynamical picture of a Bose gas, noting the importance of the chemical potential and temperature as parameters which define the equilibrium state of a thermal bath and lower-energy modes describing the condensate (and those modes affected by its presence).

We explored typical trapping potentials seen in experiments, the mathematics to restrict condensate dynamics to one or two dimensions, and the reason why topological defects such as vortices appear in the condensates used in this thesis.

Using a common finite-temperature scheme—the stochastic Gross–Pitaevskii equation—we noted that there is a 'feeding' or dissipation of atoms from a thermal bath into (and out of) lower energy and condensate modes, governed by stochastic dynamics. We then reviewed methods for the numerical implementation of the stochastic Gross–Pitaevskii equation, including important considerations such as its dimensionless form, how to ensure the magnitude and dynamics of the noisy field are appropriately described numerically, and the effect of projecting high-energy modes out of the system.

This chapter allows one to implement finite-temperature schemes numerically, noting the importance of the chemical potential and temperature in the stochastic Gross–Pitaevskii equation. This chapter provides the framework to study chapter 6 of this thesis: *Machine Learning Thermodynamical Parameters of Bose Gases*.

Part III

Applications

4

NEURAL INITIAL VALUE PROBLEM

INITIAL VALUE PROBLEMS—a system of ordinary differential equations and corresponding initial conditions—can be used to describe many physical phenomena including those arise in classical mechanics. We have developed a novel approach to solve physics-based initial value problems using unsupervised machine learning. We propose a deep learning framework that models the dynamics of a variety of mechanical systems through neural networks. Our framework is flexible, allowing us to solve non-linear, coupled, and chaotic dynamical systems. We demonstrate the effectiveness of our approach on systems including a free particle, a particle in a gravitational field, a classical pendulum, and the Hénon–Heiles system (a system of harmonic oscillators with a non-linear perturbation, used in celestial mechanics). Our results show that deep neural networks can successfully

approximate solutions to these problems, producing trajectories which conserve physical properties such as energy and those with stationary action. We note that probabilistic functions (see § 2.2.5) are *required* to learn any solutions of initial value problems in their strictest sense. We use coupled neural networks to learn solutions of coupled systems.

4.1 Statement of the physical problem

Consider a mechanical system characterised by m generalised coordinates $q_i(t)$, $i \in \{1, \cdots, m\}$, and described by the Lagrangian, L. Hamilton's principle states that the motion of a system between two times, t_1 and t_2, is such that the *action*, S, is stationary. The action is defined as the integral of the Lagrangian over time

$$S = \int_{t_1}^{t_2} \mathrm{d}t\, L(q_1, q_2, ..., q_m, \dot{q}_1, \dot{q}_2, ..., \dot{q}_m, t). \tag{4.1}$$

For the action to be stationary, its variation, δS, must vanish for small variations in the path, $q_i(t)$, i.e., $\delta S = 0$. The variation of the action is

$$\delta S = \int_{t_1}^{t_2} \mathrm{d}t\, \delta L(q_1, q_2, ..., q_m, \dot{q}_1, \dot{q}_2, ..., \dot{q}_m, t) = \int_{t_1}^{t_2} \mathrm{d}t \sum_{i=1}^{m} \left(\frac{\partial L}{\partial q_i} \delta q_i + \frac{\partial L}{\partial \dot{q}_i} \delta \dot{q}_i \right). \tag{4.2}$$

Integrating the second term by parts

$$\int_{t_1}^{t_2} \mathrm{d}t\, \frac{\partial L}{\partial \dot{q}_i} \delta \dot{q}_i = \left[\frac{\partial L}{\partial \dot{q}_i} \delta q_i \right]_{t_1}^{t_2} - \int_{t_1}^{t_2} \mathrm{d}t\, \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}_i} \right) \delta q_i. \tag{4.3}$$

Since the variation of the path vanishes at the endpoints ($\delta q_i(t_1) = \delta q_i(t_2) = 0$), the first term on the right-hand side is zero. Therefore, the variation of the action becomes

$$\delta S = \int_{t_1}^{t_2} \mathrm{d}t \sum_{i=1}^{m} \left(\frac{\partial L}{\partial q_i} - \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}_i} \right) \right) \delta q_i. \tag{4.4}$$

For δS to be zero for arbitrary variations δq_i, the integrand must vanish. Since the variations δq_i are independent, the term multiplying each δq_i must vanish individually. This gives us a set of m equations

$$\frac{\partial L}{\partial q_i} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) = 0, \quad i = 1, 2, ..., m. \tag{4.5}$$

These are the *Euler-Lagrange equations*.

We chose to consider only Lagrangians of the form $L = T - V$, where

$$T = \frac{1}{2}\sum_{i=1}^{m} \mu_i \dot{q}_i(t)^2 \tag{4.6}$$

is the kinetic energy and $V \equiv V(q_1(t), q_2(t), \cdots, q_m(t))$ is a potential energy. We consider only potential energies that are explicit functions of the generalised coordinates, which excludes time-dependent potentials and the Lorentz force. The Euler–Lagrange equations lead to equations of motion of the form

$$\mu_i \ddot{q}_i(t) - F_i(q_1(t), q_2(t), \cdots, q_m(t)) = 0, \tag{4.7}$$

where μ_i are generalised coefficients—these may be associated with a mass or moment of inertia, depending upon the dynamical system—and q_i are the generalised coordinates—which may be associated with positions or angles or other suitable coordinates that uniquely define the configuration of the system. Their dimensions will depend on the nature of each coordinate (e.g., an angle coordinate will be dimensionless, whereas a length coordinate will have dimensions of length). We can associate

$$F_i(q_1(t), q_2(t), \cdots, q_m(t)) = -\frac{\partial V(q_1(t), q_2(t), \cdots, q_m(t))}{\partial q_i(t)}, \tag{4.8}$$

with a generalised force. In order to eliminate the explicit dependence upon μ_i, let the rescaled generalised force (which may have units of acceleration or angular acceleration, as appropriate) be $f_i = F_i/\mu_i$. This leads to the equations of motion

$$\ddot{q}_i(t) - f_i(q_1(t), q_2(t), \cdots, q_m(t)) = 0, \tag{4.9}$$

where we choose to cast the equations of motion in such a way that they equal zero. This simplifies the subsequent development of the machine learning problem in § 4.2.

The domain of the problem is $t \in [0, T]$ (the time origin is set to zero without loss of generality). The problem is subject to $2m$ initial conditions, where each generalised coordinate is associated with an initial position $s_{i,0}$ and velocity $v_{i,0}$.

4.2 Statement of the machine learning problem

We will denote the neural network's representation of the solution by $\mathbf{q}_i(\mathbf{t}) = \left(q_{i,0}, q_{i,1}, \cdots, q_{i,n}, \cdots, q_{i,N_T}\right)$. Each element of the neural representation, $q_{i,n}$, is associated with the continuous solution at a particular time $t_n = n\Delta t$, i.e., $q_{i,n} \approx q_i(n\Delta t)$, where $\Delta t = T/N_T$ is the time step. We are seeking discretised vector representations $\{\mathbf{q}_i(\mathbf{t})\}$ of the continuous functions $\{q_i(t)\}$ over the temporal domain $\mathbf{t} = (0, t_1, \cdots, t_n, \cdots, t_{N_T})$.

The machine learning problem is to find an optimised neural network representation which approximates the solution of the initial value problem by minimising an appropriate cost metric. Our cost metric will be a dimensionless function (to not be dependent upon any particular system of units). We will associate all timescales with a characteristic time τ and all length scales with a characteristic length λ_i. This implies a dimensionless generalised position coordinate $\tilde{q}_i = q_i/\lambda_i$ and a dimensionless rescaled generalised force $\tilde{f}_i = f_i\tau^2/\lambda_i$. Furthermore, the dimensionless parameters α_i, β_i and γ_i may be introduced to adjust the relative weighting of each

term in the cost function. Dropping all tildes, the dimensionless cost function we wish to minimise is

$$\begin{aligned}\mathcal{C} = \sum_{i=1}^{m} \frac{1}{\lambda_i^2}\Big[\alpha_i \tau^4 \|\ddot{\mathbf{q}}_i(\mathbf{t}) - \mathbf{f}_i[\mathbf{q}_1(\mathbf{t}), \mathbf{q}_2(\mathbf{t}), \cdots, \mathbf{q}_m(\mathbf{t})]\|^2 + \\ + \beta_i \tau^2 \big[\dot{q}_{i,0} - v_{i,0}\big]^2 + \gamma_i \big[q_{i,0} - s_{i,0}\big]^2 \Big],\end{aligned} \tag{4.10}$$

where $\mathcal{C}$ produces, in general, a positive, real and scalar output. We define, in general, the notation

$$\|\mathbf{g}(\mathbf{t})\|^2 \equiv \frac{\Delta t}{\tau} \sum_{n=0}^{N_T} g(t_n)^2 = \frac{T}{\tau N_T} \sum_{n=0}^{N_T} g(t_n)^2, \tag{4.11}$$

such that

$$\lim_{\Delta t \to 0} \|\mathbf{g}(\mathbf{t})\|^2 = \frac{1}{\tau} \int_0^T \mathrm{d}t\, g(t)^2. \tag{4.12}$$

Scaling the generalised coordinates, q_i, and time, t, to be expressed in terms of λ_i and τ, respectively, ensures a dimensionless form; this corresponds to setting $\lambda_i = \tau = 1$, which we do in every example in this chapter. Furthermore, we always set $\alpha_i = \beta_i = \gamma_i = 1$. Under these assumptions, the cost function takes the form

$$\mathcal{C} = \sum_{i=1}^{m} \Big\{ \|\ddot{\mathbf{q}}_i(\mathbf{t}) - \mathbf{f}_i[\mathbf{q}_1(\mathbf{t}), \mathbf{q}_2(\mathbf{t}), \cdots, \mathbf{q}_m(\mathbf{t})]\|^2 + \big[\dot{q}_{i,0} - v_{i,0}\big]^2 + \big[q_{i,0} - s_{i,0}\big]^2 \Big\} \tag{4.13}$$

This equation states that residual of the differential equation should equal zero and constrains $\dot{q}_i(0)$ and $q_i(0)$ to not deviate significantly from the initial velocity, $v_{i,0}$, and the initial position, $s_{i,0}$. The cost function, in principle, fully specifies the initial value problem, and when the differential equation and initial conditions are satisfied, the neural network solution should tend towards the expected solution of the initial value problem.

4.3 Linear neural network representation of initial value problems

4.3.1 Linear neural network for a free particle

Statement of the equation of motion and cost function

Consider a system with the free particle Lagrangian

$$L = \frac{1}{2}\mu\dot{q}(t)^2. \tag{4.14}$$

Solving the Euler–Lagrange equations leads to the equation of motion

$$\ddot{q}(t) = 0, \tag{4.15}$$

subject to the initial position s_0 and initial velocity v_0. Equation (4.15) has solutions $q(t) = v_0 t + s_0$.

For this dynamical system, we seek neural solutions

$$\mathbf{q}(\mathbf{t}) = w\mathbf{t} + b\mathbb{1}, \tag{4.16}$$

where $\mathbb{1} \in \mathbb{R}^{N_T}$ is a vector of ones.

Figure 4.1 represents the discretised solution in equation (4.16) as a neural network. In this chapter, and in contrast to previous discussions in part II, chapter 2, neurons are always vectors—this choice means that there are fewer weights and biases to learn and that a direct equivalency of equation (4.16) with linear regression may be observed. Every neuron-neuron connection is still associated with a weight, and every neuron (in all layers except the input) is still associated with a bias. The free particle system is described entirely by the simplest conceivable neural network: it contains no hidden layers or activation functions. We define the zeroth layer neuron value as $\mathbf{t}$ and the output layer neuron as $\mathbf{q}(\mathbf{t})$. The identification of v_0 with the

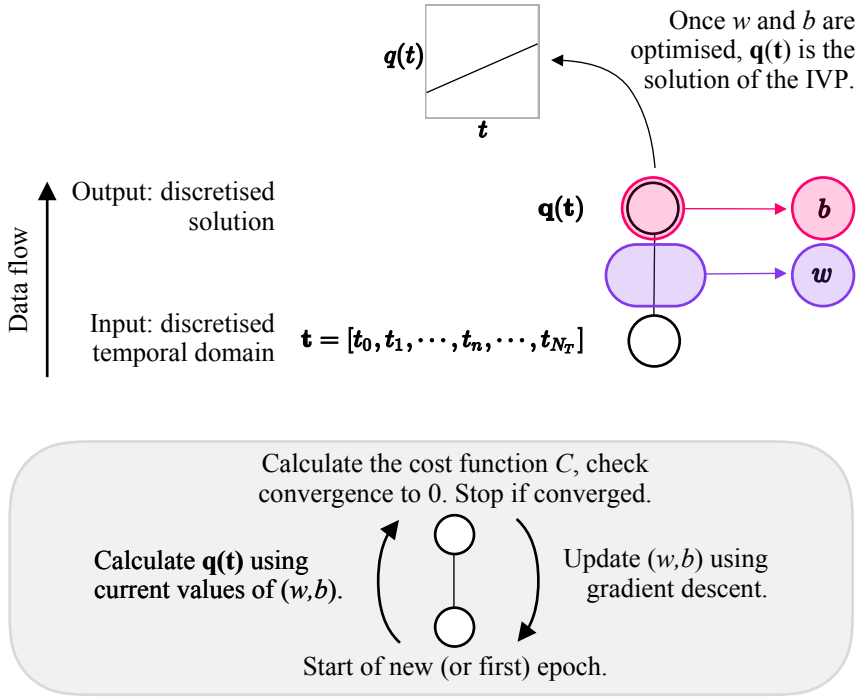


Figure 4.1: A neural network to solve the differential equation (4.15), as represented by equation (4.16) is shown. The network contains two neurons: i) an input neuron, which represents the discretised temporal domain, and ii) an output neuron, which represents the discretised solution to the initial value problem. The network consists of two parameters: a weight between the input and output vector and a bias associated with the output parameter.

weight w and s_0 with the bias is trivial, i.e., $q(0) = b$, and $\dot{q}(t) = \dot{q}(0) = w$. We can, therefore, describe the cost function as a function of the weight and bias,

$$\begin{aligned} \mathcal{C}(w,b) &= \|f(w\mathbf{t} - b\mathbb{1})\|^2 + (w - v_0)^2 + (b - s_0)^2 \\ &= \frac{T}{N_T}\sum_{n=0}^{N_T} f(wt_n + b)^2 + (w - v_0)^2 + (b - s_0)^2. \end{aligned} \tag{4.17}$$

Given that $\ddot{q} = 0 \implies f(\cdot) = 0$, the cost function reduces to

$$\mathcal{C}(w,b) = (w - v_0)^2 + (b - s_0)^2, \tag{4.18}$$

which is minimised by $w = v_0$ and $b = s_0$.

Initialisation

All parameters in any machine learning model are initialised randomly. This initialisation has no physical interpretation; it is part of the computational methodology, as explored in § 2.3. In this example, the weight and bias for the neural network are sampled from the uniform distribution

$$w, b \sim U(-1, 1). \tag{4.19}$$

The range of initial values of the weight and bias and their convergence to the expected parameters, $v_{i,0}$ and $s_{i,0}$, for 250 iterations of the training procedure is shown in Figure 4.2.

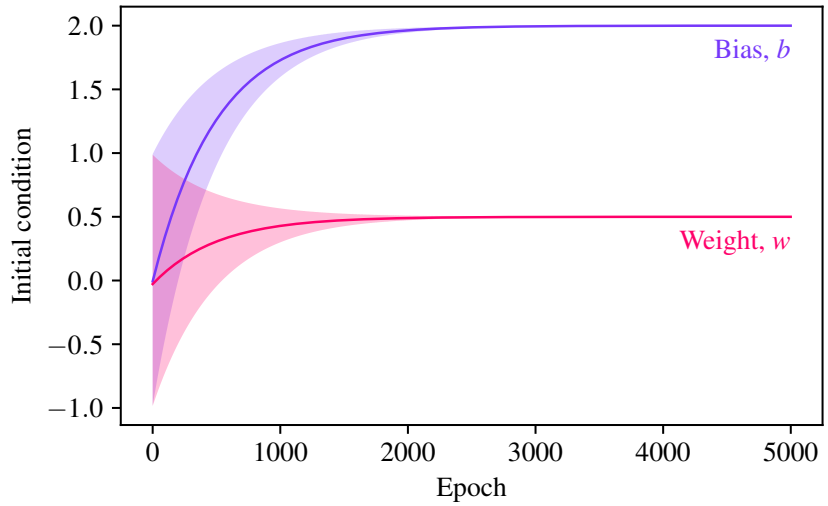


Figure 4.2: *Free particle:* The convergence of the weight and bias for an initial value problem describing a free particle with $s_0 = 2.0$ and $v_0 = 0.5$ is shown for 250 instances of the initialisation procedure. The weight and bias are initialised randomly according to equation (4.19) and tend towards their expected values after sufficient iterations of an optimisation algorithm have passed. After around 2,500 epochs, the training may be stopped as an approximate solution has been obtained. The shaded region indicates the maximum and minimum value of both parameters at a given epoch, and the line indicates the average value of the parameters throughout (which is an arithmetic mean for the underlying uniform distribution).

Learning the optimal parameters using gradient descent

Optimisation of a two parameter model can be achieved by gradient descent (usually attributed to Cauchy). Let the model parameters be denoted by the tuple (w, b). We wish to adjust the weight and bias by some Δw and Δb, respectively, where the adjustment is assumed to be small to avoid overshooting the global minima. Let the adjustments be described by the tuple $(\Delta w, \Delta b)$. Ensuring a descent in the direction of the negative gradient, the result of the first order expansion of the cost function about $(\Delta w, \Delta b)$ is

$$(\Delta w, \Delta b) = -\eta\left(\frac{\partial \mathcal{C}(w,b)}{\partial w}\Big|_{w_\mathcal{E}}, \frac{\partial \mathcal{C}(w,b)}{\partial b}\Big|_{b_\mathcal{E}}\right) \tag{4.20}$$

where $\eta > 0$ is a fixed global hyperparameter (a parameter which describes a neural network) known as the step size or learning rate and $w_\mathcal{E}$ and $b_\mathcal{E}$ denote the value of the weight and bias at epoch $\mathcal{E}$. Equation (4.20) is the epoch-to-epoch update formula for the weight and the bias. From the original definition of the cost function in equation (4.18),

$$\frac{\partial C(w,b)}{\partial w} = 2(w - v_0) \quad \text{and} \quad \frac{\partial C(w,b)}{\partial b} = 2(b - s_0). \tag{4.21}$$

Equation (4.20) describes the gradient descent throughout the parameter space. One set of parameter updates is associated with one epoch (one complete forward and backwards pass through the neural network). After a sufficient number of epochs have been completed, the weight and bias should tend toward the dimensionless initial velocity and initial position, respectively. Figure 4.3 shows the gradient descent of the weight and biases throughout the parameter space, updating according to equation (4.20).

Figure 4.4 demonstrates the neural solution given by equation (4.16) for the learnt parameters w and b. Given that the values of s_0 and v_0 obtained by the neural

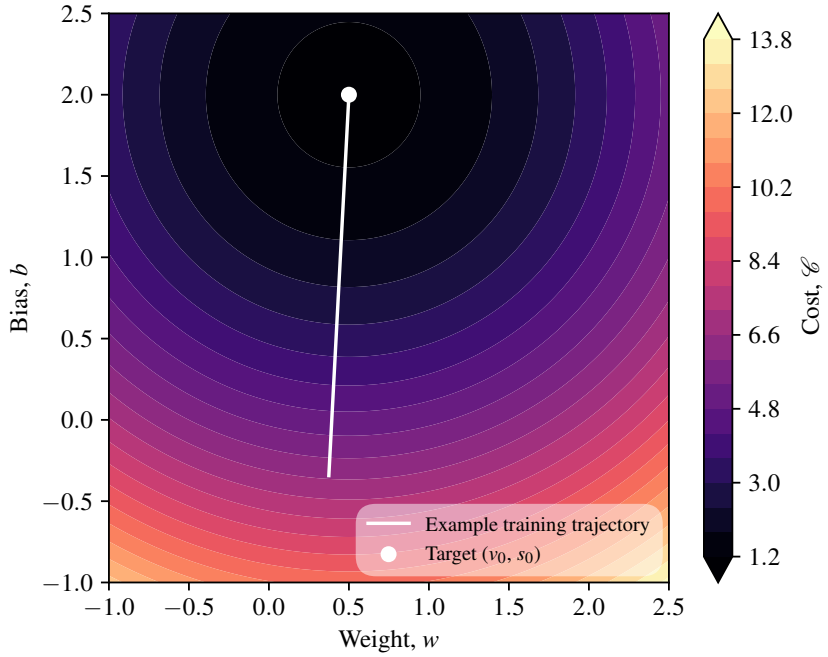


Figure 4.3: The gradient descent/update procedure for the weights and biases towards the target parameters v_0 and s_0 are shown, according to equation (4.20). The contours represent the values of the cost function in equation (4.18).

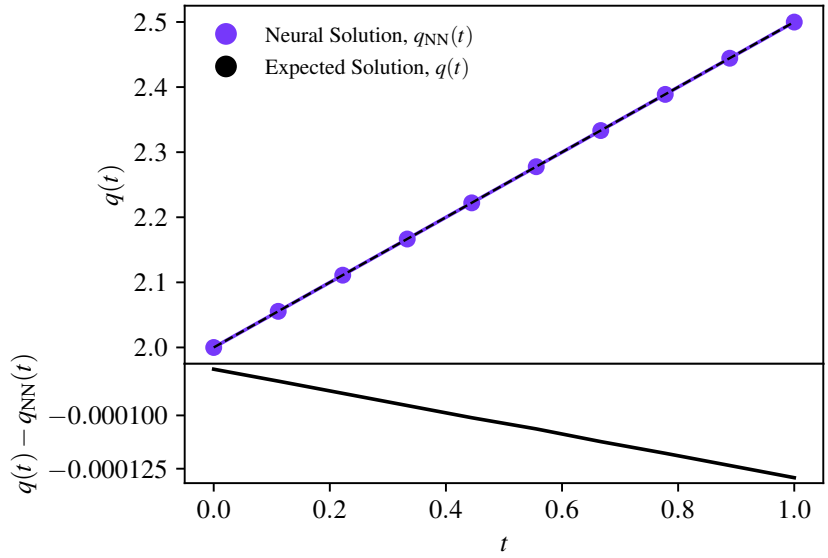


Figure 4.4: *Free particle:* The expected solution, the neural solution, and the difference (residual) of the two are shown.

network are approximations, the solution they describe will always diverge from the expected solution after sufficient time, as shown by the residual plot.

4.3.2 Linear neural network as a linear approximant for a particle in a gravitational field

This framework additionally serves as a linear approximant for functions which exhibit non-linear solutions. Consider a particle in a uniform gravitational field given by the dimensionless equation of motion

$$\ddot{q}(t) + 1 = 0, \tag{4.22}$$

which—in the linear neural network framework—continues to have solutions of the form in equation (4.16). The cost function is again a function of the weight and bias, with a non-zero forcing term

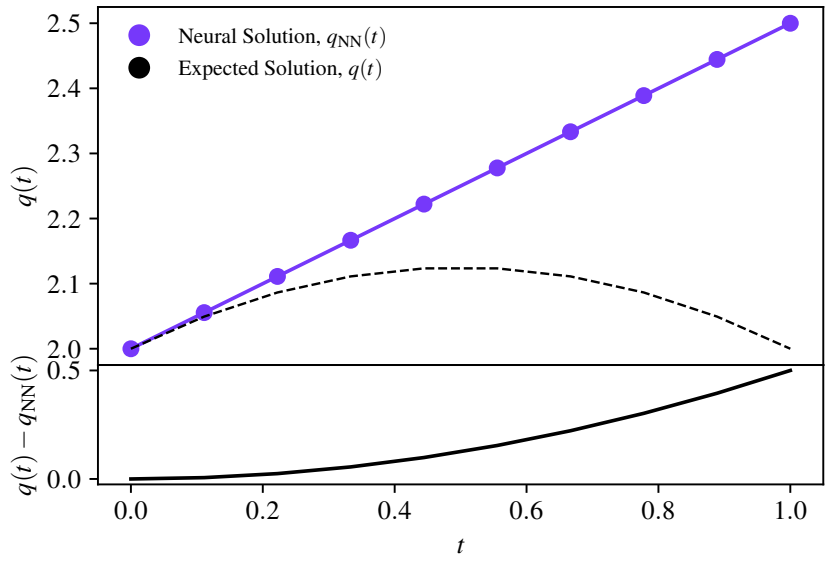
$$\begin{aligned} \mathcal{C}(w,b) &= \frac{T}{N_T}\sum_{n=0}^{N_T} 1^2 + (w - v_0)^2 + (b - s_0)^2 \\ &= \frac{T}{N_T}(N_T + 1) + (w - v_0)^2 + (b - s_0)^2 \\ &\approx T + (w - v_0)^2 + (b - s_0)^2 \end{aligned} \tag{4.23}$$

as $N_T \to \infty$. The cost function is again minimised by w and b but increases without bound as $T \to \infty$. Figure 4.5 demonstrates the linear neural network's ability to perform as a linear approximant for non-linear solutions.

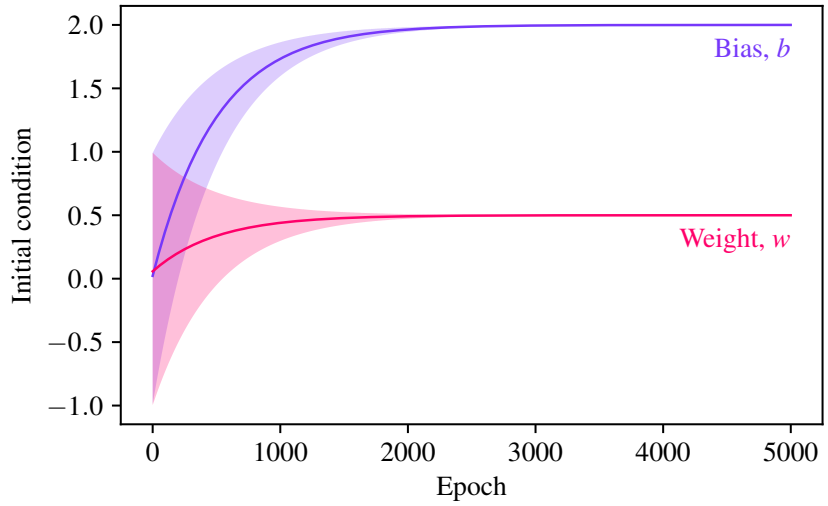
4.3.3 Linear neural network as a linear approximant for a harmonic oscillator

Consider now the harmonic oscillator given by the equation of motion

$$\ddot{q}(t) + q(t) = 0. \tag{4.24}$$



(a)



(b)

Figure 4.5: (a) The linear approximant to equation (4.16) for a particle in a gravitational field obtained by machine learning is shown alongside the expected particle trajectory. Since we are trying to fit a linear function to a quadratic function, the residuals show an underlying quadratic pattern as time evolves. (b) The average convergence of the bias and weight for 250 iterations of the training algorithm is again shown. The shaded region indicates the maximum and minimum value of both parameters at a given epoch, and the line indicates the average value of the parameters throughout (which is an arithmetic mean for the underlying uniform distribution).

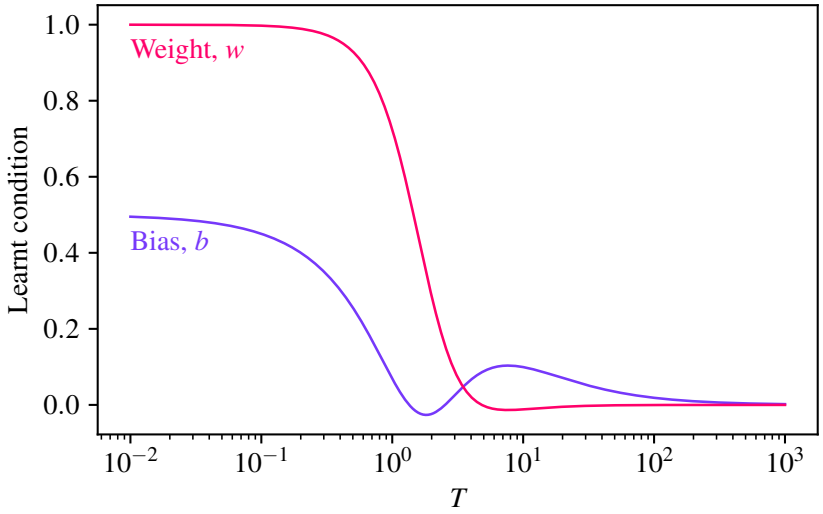


Figure 4.6: Theoretical values of the weight (equation (4.28)) and bias (equation (4.29)) describing the linear approximant of the harmonic oscillator are shown. The weight and bias are expected to tend towards zero as the time, T, increases, regardless of the true values of the initial conditions.

Letting $f(t) = -q(t) = -wt - b$, the cost function is

$$\begin{aligned} \mathcal{C}(w,b) &= \frac{T}{N_T}\sum_{n=0}^{N_T}[w(t_n)+b]^2 + (w-v_0)^2 + (b-s_0)^2 \\ &\approx \int_0^T \mathrm{d}t\,\{(wt+b)^2\} + (w-v_0)^2 + (b-s_0)^2 \\ &= \frac{w^2}{3}(T-t_0)^2 + wb(T-t_0)^2 + b^2(T-t_0) + \\ &\quad + (w-v_0)^2 + (b-s_0)^2, \end{aligned} \tag{4.25}$$

which is minimised by

$$\frac{\partial \mathcal{C}}{\partial w} = \frac{2}{3}w(T-t_0)^3 + b(T-t_0)^2 + 2(w-v_0) = 0, \tag{4.26}$$

and

$$\frac{\partial \mathcal{C}}{\partial b} = w(T-t_0)^2 + 2b(T-t_0) + 2(b-s_0) = 0. \tag{4.27}$$

Solving equations (4.26) and (4.27) simultaneously gives the weight and bias in

terms of the initial conditions v_0 and s_0 only,

$$w = \frac{-6T^2 s_0 + 12T v_0 + 12 v_0}{T^4 + 12T + 4T^3 + 12}, \tag{4.28}$$

and

$$b = \frac{4T^3 s_0 - 6T^2 v_0 + 12 s_0}{T^4 + 12T + 4T^3 + 12}. \tag{4.29}$$

It follows that, as $T \to 0$, $b = s_0$ and $w = v_0$. Furthermore, as $T \to \infty$, $b \to 0$ and $w \to 0$. That is to say, when probing long dynamical regimes as T grows larger, the linear approximant suggests that the learnt weight and bias should always tend towards zero regardless of the true values of the initial conditions. We did not observe this, and we did not explore this further. This may suggest the need for increasing the representational capacity of the neural network.

Figure 4.6 shows the values of the learnt conditions as functions of T for $(s_0, v_0) = (0.5, 1.0)$.

Learning linear solutions or linear approximations to solutions is both computationally inexpensive and easy to follow mathematically. The cost function in equation (4.18) is entirely analogous to linear regression in two dimensions (since $\ddot{q}(t) = 0$, we are only finding two optimal parameters).

4.4 Deep neural network representation of the classical pendulum

4.4.1 Statement of the equation of motion and cost function

Consider a one-dimensional pendulum with kinetic energy

$$T(\dot{\varphi}(t)) = \frac{1}{2}\mu \ell^2 \dot{\varphi}(t)^2, \tag{4.30}$$

where μ is the mass of the pendulum bob, ℓ is the length of the pendulum, and $\dot{\varphi}(t)$ is the angular velocity of the pendulum, associated with the angle coordinate $\varphi(t)$.

The potential energy is $V(\varphi(t)) = \mu g\ell[1 - \cos\varphi(t)]$, where g is the acceleration due to gravity. The Lagrangian is therefore

$$L(\varphi(t)) = \frac{1}{2}\mu\ell^2\dot{\varphi}(t)^2 - \mu g\ell[1 - \cos\varphi(t)]. \tag{4.31}$$

Introducing the characteristic time $\tau = \sqrt{\ell/g}$, and solving the Euler–Lagrange equation, the dimensionless equation of motion is $\ddot{\varphi}(t) + \sin\varphi(t) = 0$, subject to the pendulum's initial angular velocity v_0 and initial angle s_0.

The cost function for this problem is

$$\mathcal{C} = \|\ddot{\boldsymbol{\varphi}}(\mathbf{t}) - \sin(\boldsymbol{\varphi}(\mathbf{t}))\|^2 + (\dot{\varphi}_0 - v_0)^2 + (\varphi_0 - s_0)^2, \tag{4.32}$$

where $\dot{\varphi}_0$ and φ_0 are the neural network's current values of the initial angular velocity and initial position. The cost function is minimised by an appropriate set of weights and biases, which we will determine soon.

4.4.2 Towards deep learning

Unlike the linear neural network, there is no direct physical interpretation of the weights and biases other than in the small region where the solution behaves linearly. For initial value problems where their solutions exhibit non-linear behaviour, we observe that neural networks require of the order of 10^4 or 10^5 weights and biases to approximate their solution. We will use a similar architecture to Figure 4.1 where the input neuron represents the discretised temporal domain, $\mathbf{t}$, and the output neuron represents the discretised solution, $\boldsymbol{\varphi}(\mathbf{t})$. To increase the representational capacity of the neural network, we construct a multi-layer, multi-neuron network as shown in Figure 4.7. Every neuron-neuron connection is associated with a weight parameter and every neuron is associated with a bias parameter. Section 2.2 provides further details of the output of a scalar-valued neuron.

The layers between the input and output layers are called hidden layers. Computing successive hidden linear layers would be equivalent to just one linear operation—

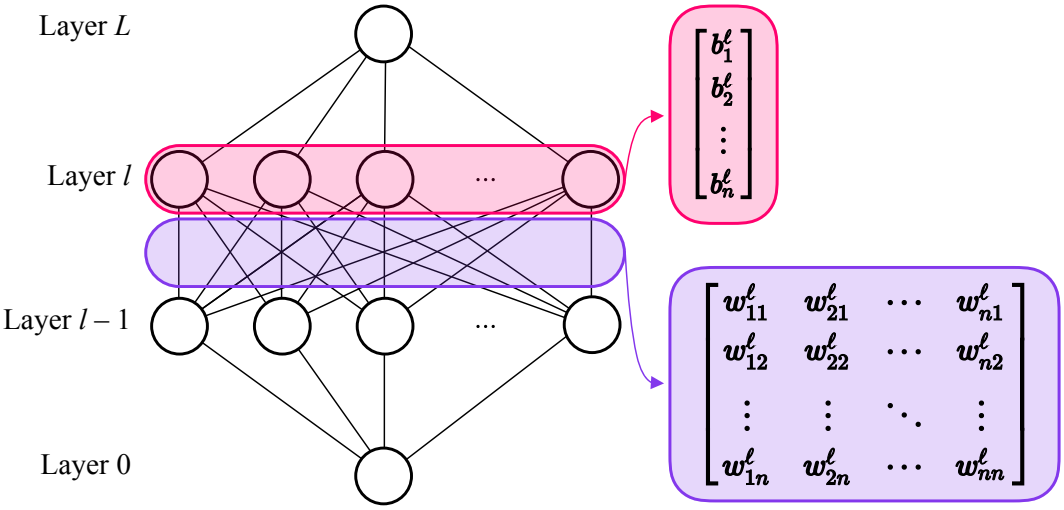


Figure 4.7: A fully connected and feedforward deep neural network is shown. A weights matrix is shown between the ℓth and $(\ell-1)$th layer, assuming that both layers contain n neurons. Every connection between every neuron is associated with a weight. A bias vector is shown on the ℓth layer—every neuron is associated with a scalar bias. The number of layers and neurons in each layer are hyperparameters which may need to be modified for different dynamical systems.

everything collapses into a linear model so that no non-linear problem can be learnt. To avoid this, we apply a non-linear function $\mathcal{A}$ to every neuron; this is referred to as an activation function, and we exploit the probabilistic interpretation of certain activation functions (see § 2.2.5) to introduce heuristically a form of model averaging (see § 2.2.5) to our solutions. For every neural network describing an initial value problem that we explore in this section, probabilistic activation functions are a fundamental component of the machine learning architecture—non-probabilistic activation functions are not conducive to learning in this context.

We assume that there are n neurons in each hidden layer (in general, this need not be the case), where each neuron has an index $j \in \{1, n\}$. We further assume that the

network has a total of L layers (excluding the input layer, which may be considered the $\ell = 0$th layer). Then, the value of a hidden neuron in the first layer is

$$\begin{aligned} \mathbf{h}_k^1 &= \mathcal{A}(\mathbf{z}_k^1), \\ \text{where} \quad \mathbf{z}_k^1 &= w_{k1}^1 \mathbf{t} + b_k^1 \mathbb{1}, \end{aligned} \tag{4.33}$$

and $\mathbb{1} \in \mathbb{R}^{N_T}$ is a vector of ones. In the subsequent layer(s), $\ell \in \{2, \cdots, L-1\}$, we similarly have

$$\begin{aligned} \mathbf{h}_j^\ell &= \mathcal{A}(\mathbf{z}_j^\ell), \\ \text{where} \quad \mathbf{z}_j^\ell &= \sum_{k=1}^{n} w_{jk}^\ell \mathbf{h}_k^{\ell-1} + b_j^\ell \mathbb{1}. \end{aligned} \tag{4.34}$$

The final (output) layer, $\ell = L$, of the neural network contains a single neuron with value

$$\begin{aligned} \boldsymbol{\varphi}(\mathbf{t}) &= \mathcal{A}(\mathbf{z}_1^L), \\ \text{where} \quad \mathbf{z}_1^L &= \sum_{k=1}^{n} w_{1j}^L \mathbf{h}_j^{L-1} + b_1^L \mathbb{1}. \end{aligned} \tag{4.35}$$

We find that $L = 3$ layer neural networks appear to be sufficient to learn the solutions to initial value problems, including the classical pendulum. Figure 4.8 shows the complete neural network representation of the classical pendulum in a three-layer architecture (with two hidden layers and one output layer), alongside Eqs. (4.33–4.35).

4.4.3 Initialisation

We initialise the weights and biases in the hidden layers from the uniform distribution

$$w_{jk}^\ell, b_j^\ell \sim U\left(-\sqrt{\frac{3}{n}}, +\sqrt{\frac{3}{n}}\right). \tag{4.36}$$

See § 2.3 for more details.

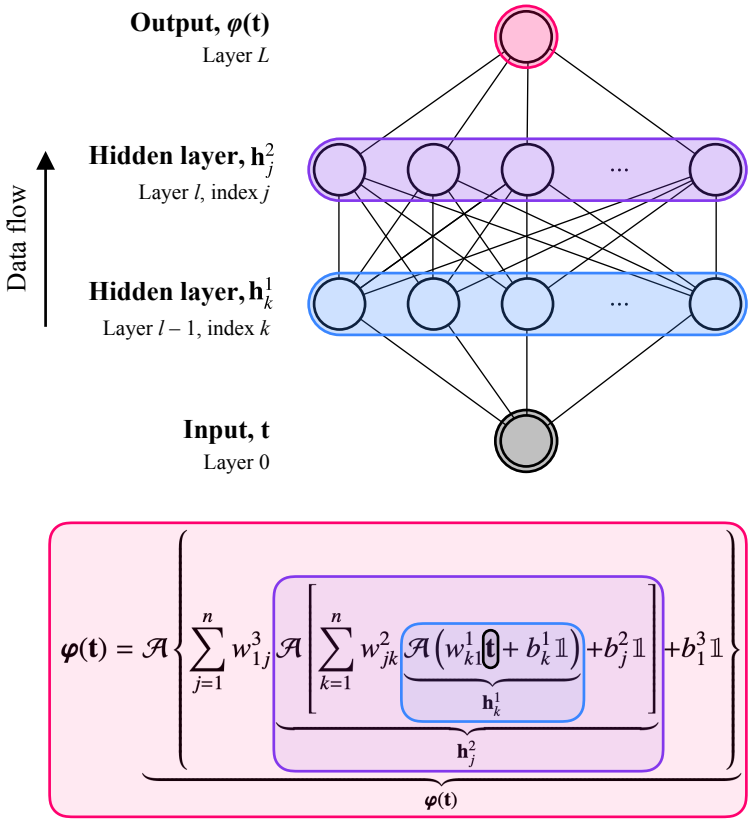


Figure 4.8: The neural network in Figure 4.7 is annotated to show which layers compute Eqs. (4.33–4.35). It demonstrates that the neural network is essentially a composition of many functions. A discretised temporal domain $\mathbf{t}$ with N_T time steps is input to the network. The discretised solution $\boldsymbol{\varphi}(\mathbf{t})$ is output. Information flows from bottom to top. Every neuron is associated with a tensor of dimensions $[1, N_T, 1]$. Every hidden layer is a tensor of shape $[1, N_T, n]$.

4.4.4 *Learning the optimal parameters using adaptive moment estimation*

We wish to learn the cost function minimising mapping between the input layer and the output layer. Gradient descent will eventually reach a global minimum deterministically, although it may take a very long time to traverse flat regions of the parameter space where parts of the gradient of the cost function are approximately zero. Instead, in all examples in this section, we use the adaptive moment estimation (Adam) algorithm, which we derive in § 2.5.3. The gradients of the weights and biases

are determined by an algorithm called *backpropagation*. The standard equations of backpropagation for scalar-valued neurons are presented in § 2.5.4. We will now derive the equations of backpropagation for vector-valued neurons.

4.4.5 Backwards propagation of errors with vector-valued neurons

Forward propagation computes an estimate of the vector $\boldsymbol{\varphi}(\mathbf{t})$ based on the existing weights and biases. The cost function, $\mathcal{C}$, is then calculated for that particular estimate. We want to determine the *sensitivity* of the cost function with respect to the weights and biases throughout the entire network. The sensitivities of the cost function are given by

$$\frac{\partial \mathcal{C}}{\partial w_{jk}^{\ell}} = \nabla_{\mathbf{z}_j^{\ell}} \mathcal{C} \cdot \nabla_{w_{jk}^{\ell}} \mathbf{z}_j^{\ell} \tag{4.37a}$$

$$\frac{\partial \mathcal{C}}{\partial b_j^{\ell}} = \nabla_{\mathbf{z}_j^{\ell}} \mathcal{C} \cdot \nabla_{b_j^{\ell}} \mathbf{z}_j^{\ell} = \nabla_{\mathbf{z}_j^{\ell}} \mathcal{C} \cdot \mathbb{1} = \nabla_{\mathbf{z}_j^{\ell}} \mathcal{C}. \tag{4.37b}$$

which we obtain by the chain rule. We use $\cdot$ to denote the scalar product for notational simplicity. Using the definition of the output of a neuron in the first hidden layer in Eq. (4.33) and in an arbitrary layer, ℓ, in Eq. (4.34), we can see that the sensitivity of the cost function with respect to the network weights is

$$\frac{\partial \mathcal{C}}{\partial w_{jk}^{\ell}} = \begin{cases} \nabla_{\mathbf{z}_j^{\ell}} \mathcal{C} \cdot \mathbf{h}_j^{\ell}, & \text{if } \ell \geq 2, \\ \nabla_{\mathbf{z}_j^{\ell}} \mathcal{C} \cdot \mathbf{t}, & \text{if } \ell = 1. \end{cases} \tag{4.38}$$

We seek an equation for $\nabla_{\mathbf{z}_j^{\ell}} \mathcal{C}$, which is calculated backwards through the network for computational efficiency [GW08]. This is the sensitivity of the cost function with respect to the weighted input to the neuron. In the final layer, this sensitivity is given by

$$\nabla_{\mathbf{z}_1^L} \mathcal{C} = \nabla_{\boldsymbol{\varphi}} \mathcal{C} \, \nabla_{\mathbf{z}_1^L} \boldsymbol{\varphi} \,. \tag{4.39}$$

The sensitivity is, in a linear algebra sense, a Jacobian matrix (of the partial derivatives of the elements of $\boldsymbol{\varphi}$ with respect to the elements of $\mathbf{z}_1^L$) multiplied by a row vector (of the partial derivatives of $\mathcal{C}$ with respect to each element of the solution vector $\boldsymbol{\varphi}$). Since the activation function is applied element-wise, the Jacobian $\nabla_{\mathbf{z}_1^L}\boldsymbol{\varphi}$ is diagonal, so the sensitivity reduces to an element-wise multiplication, as indicated by the Hadamard product, $\odot$. It therefore follows that

$$\nabla_{\mathbf{z}_1^L}\mathcal{C} = \nabla_{\boldsymbol{\varphi}}\mathcal{C} \odot (\nabla_{\mathbf{z}_1^L} \odot \boldsymbol{\varphi}), \tag{4.40}$$

where the substitution $\boldsymbol{\varphi} = \mathcal{A}(\mathbf{z}_1^L)$ can then be made—this directly introduces the derivative of the activation function into the backpropagation algorithm.

We then work top to bottom through the neural network (hence, backpropagation) to determine the optimality of each neuron. Let us consider two arbitrary layers in the network, ℓ and $\ell - 1$, with neuron indices j and k, respectively. This preserves the previously defined ordering of indices on, e.g., the weights w_{jk}^{ℓ}. For the layer $\ell - 1$, the sensitivity of the cost function with respect to the kth neuron can be written in terms of all of the sensitivities in the layer after it (this neuron influences all neurons in the subsequent layers ℓ, $\ell + 1$, and so on, in a fully connected neural network). The sensitivity of the output of a neuron in layer $(\ell - 1)$ is therefore also determined by the chain rule

$$\nabla_{\mathbf{z}_k^{\ell-1}}\mathcal{C} = \sum_{j=1}^{n_\ell} \nabla_{\mathbf{z}_j^{\ell}}\mathcal{C}\, \nabla_{\mathbf{z}_k^{\ell-1}}\mathbf{z}_j^{\ell}, \tag{4.41}$$

where each term in the summation is a Jacobian matrix (the matrix of partial derivatives of pre-activations from one layer to the next) multiplied by a vector (the sensitivity in the cost function with respect to a change in the pre-activations in the subsequent layers).

Let the Jacobian matrix $\mathsf{J}^{jk} = \nabla_{\mathbf{z}_k^{\ell-1}} \mathbf{z}_j^{\ell}$ with elements

$$\mathsf{J}^{jk}_{mn} = \frac{\partial (z_j^{\ell})_m}{\partial (z_k^{\ell-1})_n}. \tag{4.42}$$

By the definition of the pre-activation in Eq. (4.34), it follows that each element J^{jk}_{mn} is

$$\mathsf{J}^{jk}_{mn} = \sum_{k'=1}^{n_\ell} w^{\ell}_{jk'} \frac{\partial (\mathcal{A}((z_{k'}^{\ell-1})_m))}{\partial ((z_k^{\ell-1})_n)}, \tag{4.43}$$

which can only be nonzero if $k' = k$ and $m = n$, such that the function being differentiated is a function of the variable it is being differentiated with respect to. Therefore, the matrices J^{jk} are again diagonal, with

$$\mathsf{J}^{jk}_{nn} = w^{\ell}_{jk} \frac{\partial (\mathcal{A}((z_k^{\ell-1})_n))}{\partial ((z_k^{\ell-1})_n)} \tag{4.44}$$

and $\mathsf{J}^{jk}_{m\neq n} = 0$. Substituting Eq. (4.43) into equation Eq. (4.41) gives the error in the $(\ell-1)$th layer in terms of the errors in the ℓth layer,

$$\nabla_{\mathbf{z}_k^{\ell-1}} \mathcal{C} = \sum_{j=1}^{n_\ell} w^{\ell}_{jk} \nabla_{\mathbf{z}_j^{\ell}} \mathcal{C} \odot \Big(\nabla_{\mathbf{z}_k^{\ell-1}} \odot \mathcal{A}(\mathbf{z}_k^{\ell-1}) \Big). \tag{4.45}$$

It immediately follows that between the output and the penultimate layer,

$$\nabla_{\mathbf{z}_j^{\ell}} \mathcal{C} = w^{L}_{1j} \nabla_{\mathbf{z}_1^{L}} \mathcal{C} \odot \Big(\nabla_{\mathbf{z}_j^{\ell}} \odot \mathcal{A}(\mathbf{z}_j^{\ell}) \Big). \tag{4.46}$$

4.4.6 Results and discussion

The exact solution for the pendulum dynamics, for arbitrary initial angle and angular velocity, may be written in terms of Jacobi elliptic functions, but the standard method of solution is via numerical integration schemes. There is a numerical instability when $\varphi(0) = \pi$ and $\dot{\varphi}(0) = 0$; the pendulum should remain at $\varphi(t) = \pi$ for all time,

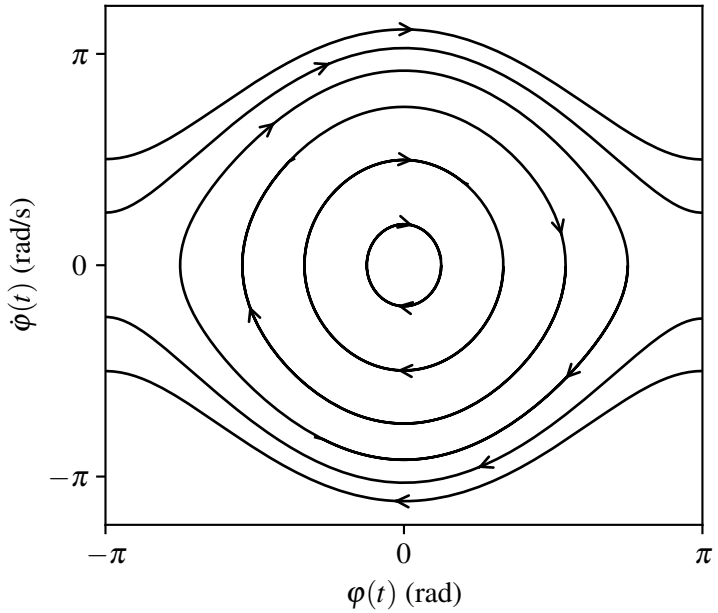


Figure 4.9: The phase portrait for four open and four closed trajectories is shown. Small artefacts can be observed (for example, on the closed trajectory with the highest initial angle) since the error in the neural solution increases as time increases. Note that the unstable trajectory at $\varphi(0) = \pi$ and $\dot{\varphi}(0) = 0$ does not appear on the phase space portrait — it only exists at a single point.

but machine precision means that the pendulum will always fall, as the pendulum is never at exactly $\varphi(0) = \pi$.

We sample both open and closed trajectories of the classical pendulum. The closed trajectories have initial positions from $s_0 = \pi/8$ to $s_0 = 3\pi/4$ — equally spaced and chosen to cover a broad range of dynamical behaviour — and zero velocity, $v_0 = 0$. The open trajectories have initial positions $s_0 = \pm\pi$ and velocities $v_0 = \pm\pi/2$ or $v_0 = \pm\pi$ (the open and positive trajectories have negative initial angle and positive initial angular velocity, and the open and negative trajectories have positive initial angle and negative initial angular velocity). All trajectories are shown in Figure 4.9. Figure 4.10 shows the various cost metrics for each trajectory. The machine learning modelling estimates the solution of the initial value problem for a given set of weights and biases, and determines the mean square error of the trajectory across the problem

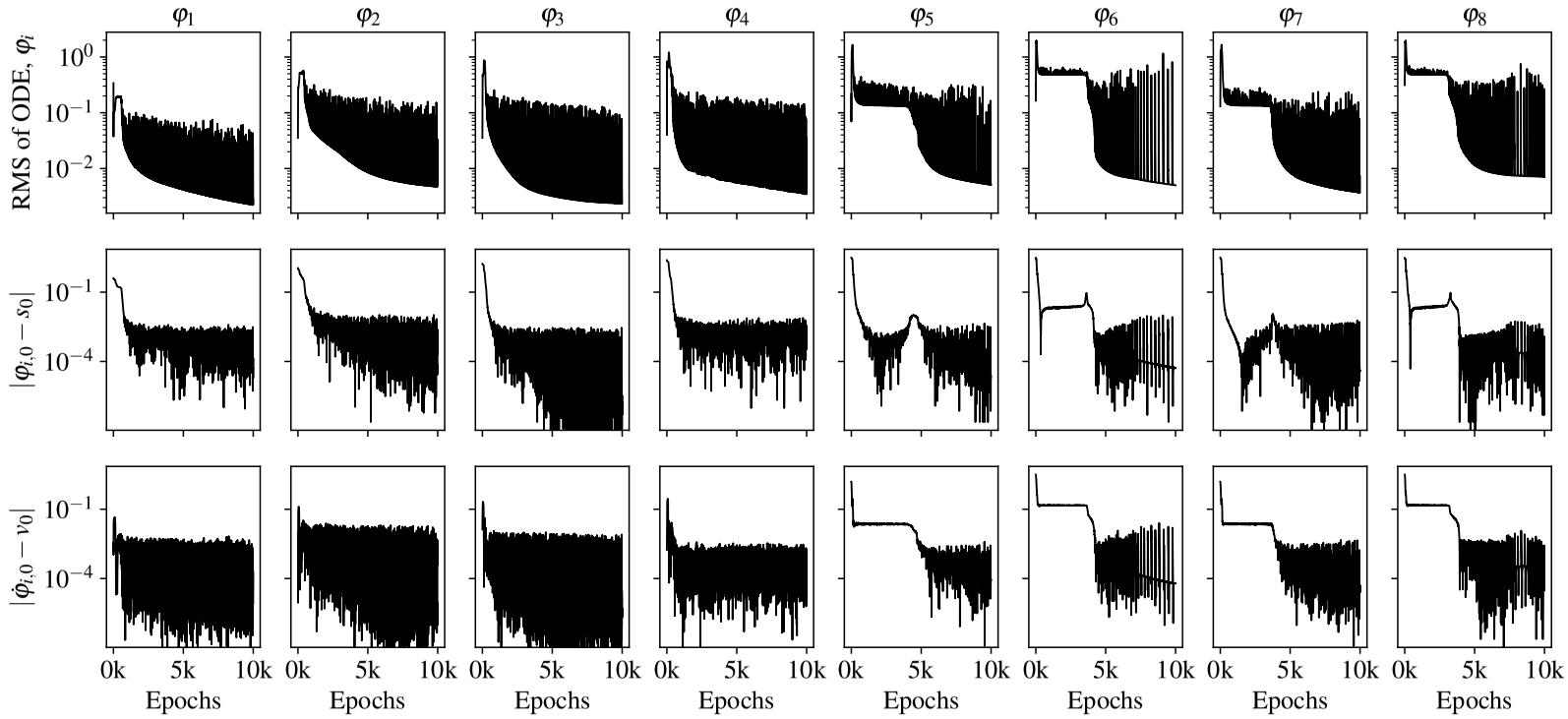


Figure 4.10: Cost metrics for all trajectories in the classical pendulum that were probed in Figure 4.9. The closed trajectories have initial angles $\varphi_{1,0} = \pi/8$, $\varphi_{2,0} = \pi/3$, $\varphi_{3,0} = 13\pi/24$ and $\varphi_{4,0} = 3\pi/4$ (these are four equally spaced angles covering a broad range of dynamical behaviour) and initial angular velocities $\dot{\varphi}_{i=1,\cdots,4} = 0$. The open, positive trajectories have initial angles $\varphi_{5,0} = -\pi$ and $\varphi_{6,0} = -\pi$ and initial angular velocities $\dot{\varphi}_{5,0} = \pi/2$ and $\dot{\varphi}_{6,0} = \pi$. The open, negative trajectories have initial angles $\varphi_{7,0} = \pi$ and $\varphi_{8,0} = \pi$ and initial angular velocities $\dot{\varphi}_{7,0} = -\pi/2$ and $\dot{\varphi}_{8,0} = -\pi$. All cost metrics use a logarithmic scale.

domain. In order to plot a quantity with physical dimensions, we take the square root of this estimate; we define the quantity $\sqrt{\|\ddot{\mathbf{q}}_i(\mathbf{t}) - \mathbf{f}_i(\mathbf{q}_1(\mathbf{t}), \mathbf{q}_2(\mathbf{t}), \cdots, \mathbf{q}_m(\mathbf{t}))\|^2}$ to be the root-mean-square of the differential equation (RMS of the ODE). Remarkably, we note that the machine learning modelling successfully captured the numerical instability at π radians, although this took around ten times more epochs than other trajectories.

4.5 Coupled neural network for the Hénon–Heiles system

4.5.1 Statement of the equation of motion and cost function

The Hénon–Heiles system [HH64] describes the non-linear motion of a star around its galactic centre. The motion can be viewed as a pair of harmonic oscillators with a perturbation (associated with a galactic potential) which couples the oscillators together. The motion is chaotic and non-dissipative. This system can, again, be described in terms of a Lagrangian of the form $L = T - V$, where the kinetic energy is

$$T(\dot{x}(t), \dot{y}(t)) = \frac{1}{2}m\big[\dot{x}^2(t) + \dot{y}^2(t)\big], \tag{4.47}$$

and the potential is

$$V(x(t), y(t)) = \frac{1}{2}m\omega^2\big[x^2(t) + y^2(t)\big] + k\left[x^2(t)y(t) - \frac{y^3(t)}{3}\right]. \tag{4.48}$$

Ensuring a dimensionless system of equations by setting $m = \omega = k = 1$, and by solving the Euler–Lagrange equations, we obtain the system of coupled equations of motion for each coordinate

$$\begin{aligned} \ddot{x}(t) + x(t) - 2x(t)y(t) &= 0, \\ \ddot{y}(t) + y(t) + y^2(t) - x^2(t) &= 0. \end{aligned} \tag{4.49}$$

The system is time-independent, and so the total energy $E = T + V$ is a conserved quantity.

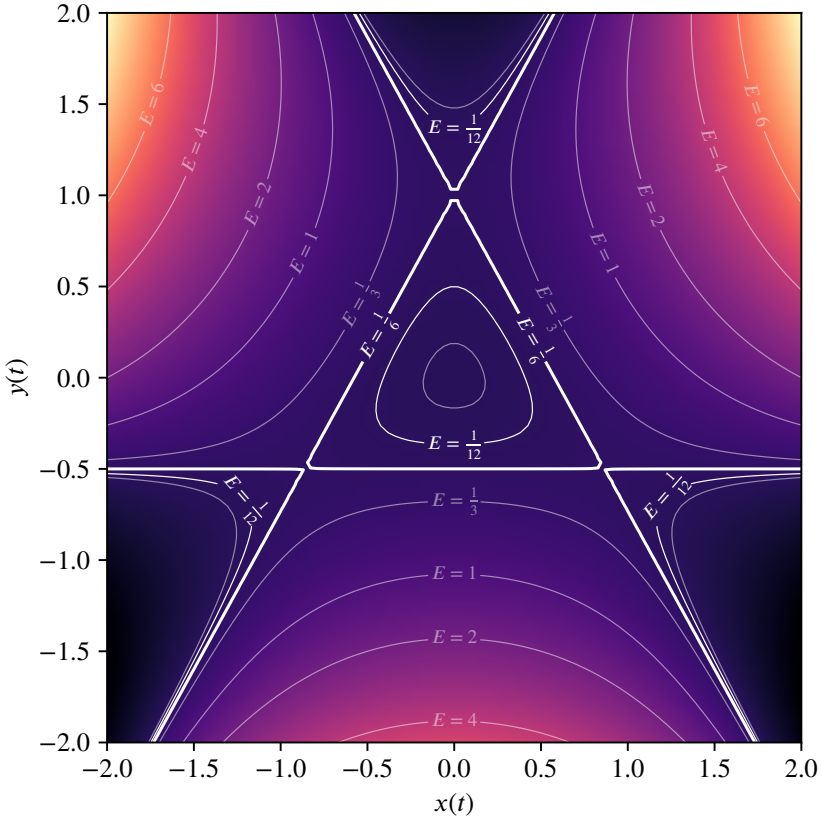


Figure 4.11: Equipotential curves for the Hénon–Heiles potential in equation (4.48) are shown for a range of energies. An equilateral triangle with points at $(-\sqrt{3/2}, -1/2)$, $(0, 1)$ and $(+\sqrt{3/2}, -1/2)$ forms the equipotential curve for $E = 1/6$, the energy below which all trajectories are bounded. Open trajectories are obtained when the energy of the particle exceeds $E = 1/6$.

The equations of motion and initial conditions lead to the cost function

$$\begin{aligned} \mathcal{C} = &\|\ddot{\mathbf{x}}(\mathbf{t}) + \mathbf{x}(\mathbf{t}) - 2\mathbf{x}(\mathbf{t})\mathbf{y}(\mathbf{t})\|^2 \\ &+ \|\ddot{\mathbf{y}}(\mathbf{t}) + \mathbf{y}(\mathbf{t}) + \mathbf{y}^2(\mathbf{t}) - \mathbf{x}^2(\mathbf{t})\|^2 \\ &+ (x_0 - s_{x,0})^2 + (y_0 - s_{y,0})^2 \\ &+ (\dot{x}_0 - v_{x,0})^2 + (\dot{y}_0 - v_{y,0})^2. \end{aligned} \tag{4.50}$$

4.5.2 Coupled neural networks

We have two dynamical degrees of freedom, which we handle by employing two neural networks — one for each coordinate — with separate parameterisations that we simultaneously optimise. The cost function is shared between the two networks, although note that there are two separate backwards propagations of errors needed to update the parameters of each network. The gradients computed during both backward passes are based on the cost function in Eq. (4.50), which includes terms for both coordinates. Because these coordinates are coupled in the equations of motion, the gradient of the cost function with respect to one set of parameters will naturally include contributions from both coordinates. The presence of coupling in the cost function means that we must ensure that the updates to the parameters for one network reflect the influence of the other network. The first backward propagation computes the gradients in the network representing x; the gradient information is retained after the errors associated with the first coordinate are computed. The second backward propagation computes the gradients in the network representing y; all gradient information is then discarded since all backwards passes through all coordinates are complete.

Once we calculate the gradients, the optimiser takes a step to update the parameters in the direction that minimises the cost function. The process of zeroing (or resetting) the gradients, calculating new gradients, and then stepping the optimiser is repeated many times in a training loop until the networks converge to a solution that minimises the cost function. In principle, the coupled optimiser could include as many optimisers as computer memory allows. We refer the reader to our code [Gri24c] for further implementation details.

Figure 4.12 shows a schematic for the coupled neural network for the Hénon–Heiles system.

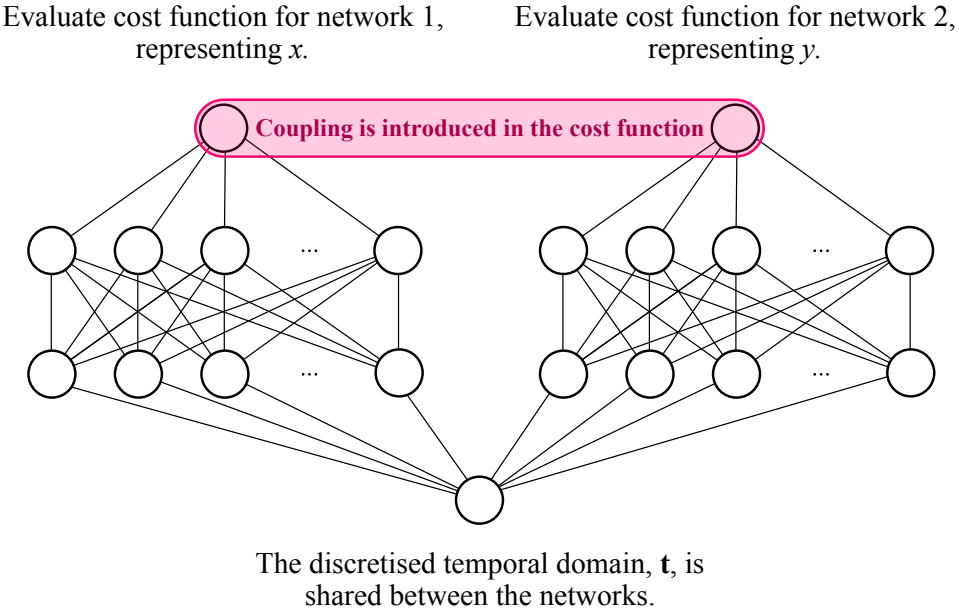


Figure 4.12: A coupled neural network for the Hénon–Heiles system. It propagates independent parameters for each generalised coordinate (given a common temporal domain) and couples only in the evaluation of the cost functions.

4.5.3 Results and discussion

We will only consider confined (closed) trajectories. Hénon and Heiles give an upper limit for the energy to produce a confined trajectory as $E = 1/6$. This trajectory is entirely chaotic. We also study the confined trajectory with energy $E = 1/12$, which is quasi-periodic. We choose always to set the initial position of the first coordinate such that $s_{x,0} = 0$. For the chaotic trajectory, where $E = 1/6$, we set the initial velocity for the x-coordinate to $v_{x,0} = 0.10$, and the initial position and velocity for the y-coordinate to $s_{y,0} = 0.57$ and $v_{y,0} = 0.057$. For the quasi-periodic trajectory,

where $E = 1/12$, we set the velocity for the x-coordinate to $v_{x,0} = 0.05$, and the initial position and velocity for the y-coordinate to $s_{y,0} = 0.40$ and $v_{y,0} = 0.041$.[1]

Figures 4.13a and 4.13b show these two Hénon–Heiles trajectories. We used the SechLU activation function, given in Eq. (2.10c). GELU, given in Eq. (2.9c), was not conducive to learning and all trajectories failed to obtain correct dynamics; we are unsure why this is the case, but it is our empirical observation that SechLU is the most reliable of the probabilistic activation functions that we have tested. Figures 4.14 and 4.15 show the individual cost metrics for a Hénon–Heiles system with energies $E = 1/6$ and $E = 1/12$, respectively.

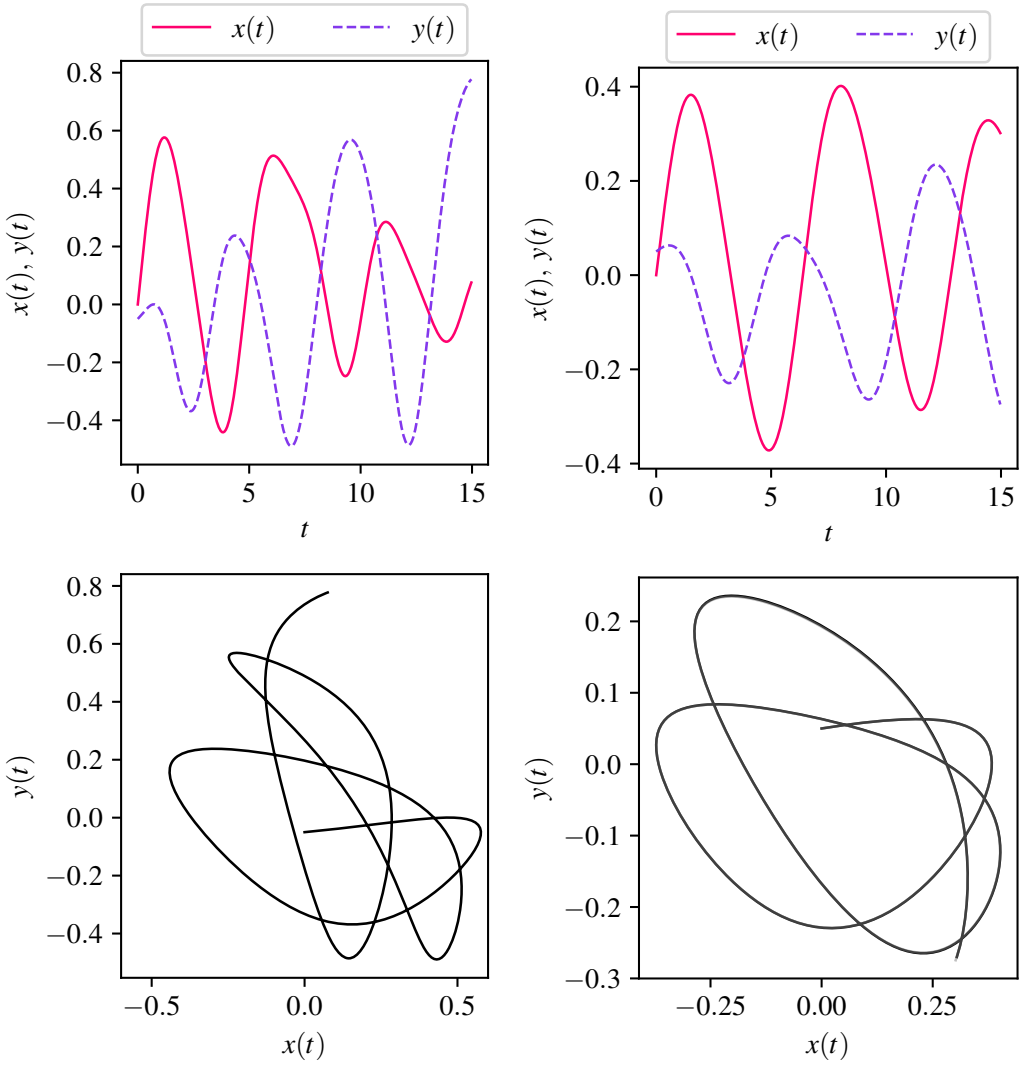
The neural solutions successfully minimise the action, S, as training progresses (we stress that the action is not an explicit term in the cost function, although it generates the dynamical equations through Hamilton's principle). We approximate the action at every epoch using $S \approx \Delta t \sum_{k=0}^{N_T} L(x_k, \dot{x}_k, y_k, \dot{y}_k)$. We note that the action of the machine learning trajectories does not necessarily increase or decrease monotonically with epoch. We note that one might expect that forming a cost function around the action only and then minimising it would be sufficient, as an almost literal realisation of Hamilton's principle of stationary action (the fact that the action must be extremised rather than strictly minimised notwithstanding). We have not found this to be the case, neither in conjunction with a trajectory's initial values, nor with coordinate values at the path's two endpoints. The Euler–Lagrange equations, i.e., the dynamical *consequence* of Hamilton's principle, in conjunction with the initial conditions, appear to be necessary, which also have the useful property of the minimising value of the cost function always being equal to zero. We note that when the root mean square of the differential equation remains constant over many epochs, it corresponds to unchanging values of the action. This

[1]The initial velocity of the first coordinate is constrained by $\dot{x}(0) = \sqrt{2E - v_{y,0}^2 - s_{y,0}^2 + 2s_{y,0}^3/3}$, where we have set either $E = 1/6$ or $E = 1/12$. Given these constraints, the initial position and initial velocity for the second coordinate can be arbitrarily chosen, providing that $\dot{x}(0)$ remains real-valued. To ensure an appropriate choice of the initial conditions for the second coordinate, we find $y(0)$ by using root finding algorithms on the constraint $V < E$, and setting the result as 10% of this value. We find $\dot{y}(0)$ by evaluating $T < E - V$, and setting the result as 10% of this value.

is due to the equivalence of the principle of stationary action and the underlying equations of motion.

The trajectories obtained by the Runge–Kutta integration routines may not give the 'true' value of the action (especially in systems with chaotic dynamics). However, since multiple methods are approaching similar values of the action, this value could be interpreted as the most stationary.

We finally note that a finer temporal discretisation appears to improve the learning capabilities of the neural network, especially with the chaotic trajectory, without an increase in network parameters (since the neurons are vector-valued). The stability of probing chaotic dynamics using numerical integration schemes such as the Runge–Kutta methods can be improved (up to a point) by using a finer temporal resolution, which is what we observe in our neural solutions.



(a) (b)

Figure 4.13: Two Hénon–Heiles trajectories obtained by machine learning with different energies: (a) corresponds to $E = 1/6$ (the chaotic trajectory) and (b) corresponds to $E = 1/12$ (the quasi-periodic trajectory). We show comparative metrics for the same trajectories obtained by Runge–Kutta methods in Figure 4.14 and Figure 4.15. **Top**: the time evolution of the individual coordinates. **Bottom**: phase profiles, showing (x, y) pairs as parametric functions of t.

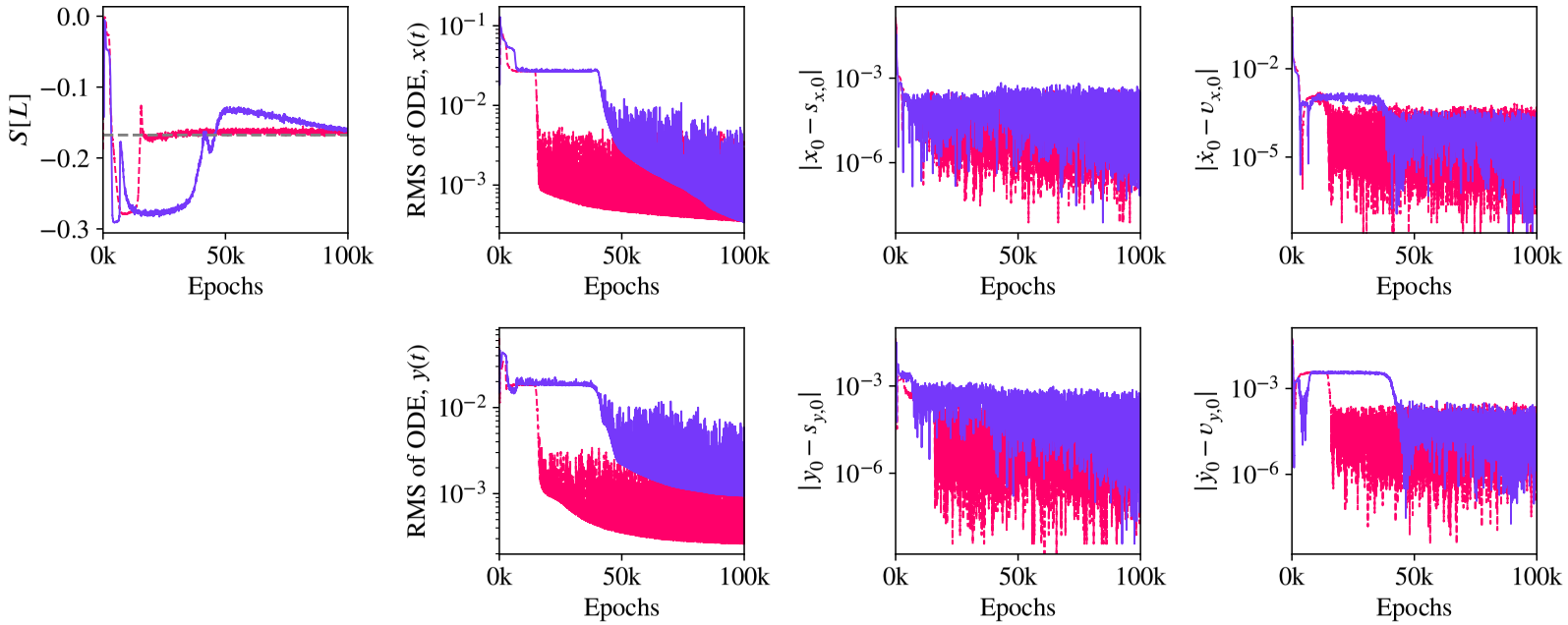


Figure 4.14: *Hénon–Heiles system for energy* $E = 1/6$. **Row 1:** the evolution of the action as training progresses. The dashed, grey horizontal lines indicate the action of the system obtained by using a Runge–Kutta 4th-order integration scheme. The dashed, pink trajectories are associated with $N_T = 512$ timesteps. The solid, purple trajectories are associated with $N_T = 256$ timesteps. As the training evolves, an extremum of the action is also found, taking similar values to the Runge–Kutta integration schemes. **Row 2:** the root mean square of the residual of the differential equations for the respective coordinates. **Row 3:** the residuals in the initial positions for both coordinates. **Row 4:** the residuals in the initial velocities for both coordinates. The initial conditions are easier to match since the solution begins to diverge as the time coordinate increases.

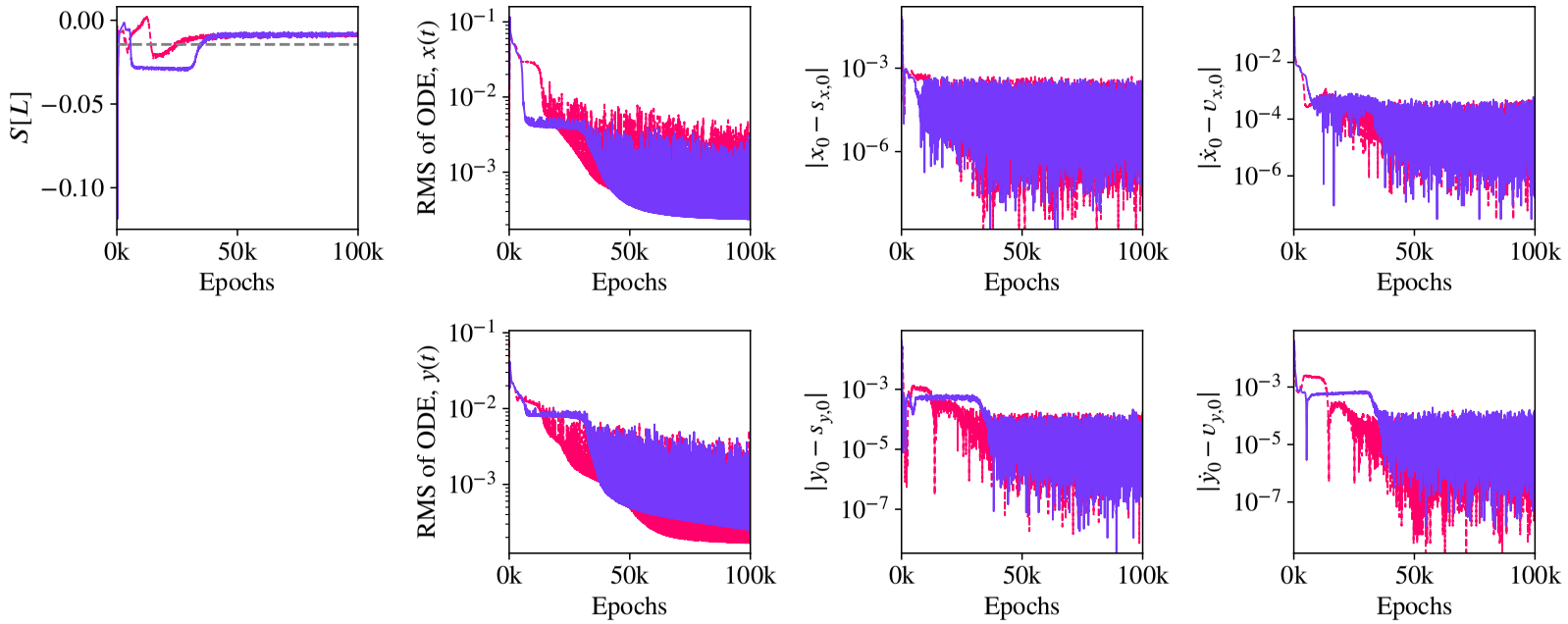


Figure 4.15: *Hénon–Heiles system for energy* $E = 1/12$. **Row 1:** the evolution of the action as training progresses. The dashed, grey horizontal lines indicate the action of the system obtained by using a Runge–Kutta 4th-order integration scheme. The dashed, pink trajectories are associated with $N_T = 512$ timesteps. The solid, purple trajectories are associated with $N_T = 256$ timesteps. As the training evolves, an extremum of the action is also found, taking similar values to the Runge–Kutta integration schemes. **Row 2:** the root mean square of the residual of the differential equations for the respective coordinates. **Row 3:** the residuals in the initial positions for both coordinates. **Row 4:** the residuals in the initial velocities for both coordinates. The initial conditions are easier to match since the solution begins to diverge as the time coordinate increases.

4.6 Neural representation of a complex-valued initial value problem

Consider a complex-valued function

$$\alpha(t) = u(t) + iv(t), \tag{4.51}$$

where $u(t), v(t) \in \mathbb{R} \to \mathbb{R}$. Consider further the first-order, complex ordinary differential equation

$$\dot{\alpha}(t) = f(t, \alpha(t)), \tag{4.52}$$

subject to the initial condition $\alpha(0) = a + ib \in \mathbb{C}$ (thus completely specifying an initial value problem), where f is a complex-valued function of the form

$$f(t, \alpha(t)) = p(t, u, v) + iq(t, u, v), \tag{4.53}$$

and where $p(t, u, v), q(t, u, v) \in \mathbb{R} \to \mathbb{R}$. Substitution of equation (4.51) and equation (4.53) into the differential equation (4.52) gives

$$\dot{u}(t) + i\dot{v}(t) = p(t, u, v) + iq(t, u, v). \tag{4.54}$$

By equating the real and imaginary parts of equation (4.54), one obtains a system of first-order, real coupled differential equations

$$\dot{u}(t) = p(t, u, v) \tag{4.55a}$$

$$\dot{v}(t) = q(t, u, v), \tag{4.55b}$$

subject to the initial conditions $u(0) = a$ and $v(0) = b$, respectively. Equation (4.55a), equation (4.55b) and these initial conditions serve as a framework for the solution of first-order, ordinary differential equations. This system of differential equations can be solved using coupled neural networks.

4.7 Summary

We have presented the neural initial value problem: a machine learning framework which approximates the solutions to a range of physics-based initial value problems in the absence of training data (i.e., an unsupervised learning approach). We have formulated a general cost function that captures the dynamics of a wide range of mechanical systems. Through extensive experimentation, we demonstrated that our approach can successfully model systems with non-linear, coupled, and chaotic behaviours. For the free particle and particle in a gravitational field, our linear neural network provided accurate solutions and served as a linear approximant for problems with non-linear solutions. In the case of the pendulum (for arbitrary angles, i.e., outside the small angle approximation), the deep neural network framework proved effective in capturing non-linear dynamics. For the Hénon–Heiles system, our coupled neural network was successful in capturing the dynamics of the two coordinates, including in chaotic regimes. We have also introduced and evaluated several machine learning techniques to enhance our framework: probabilistic activation functions, which appear to be necessary to learn the solutions of initial value problems (they may be interpreted as a general class of model averaging functions), and the coupled optimisation routines allow for the simultaneous optimisation of many neural networks representing individual generalised coordinates.

Whilst not a direct replacement for standard numerical integration routines such as the Runge–Kutta methods (which, in our benchmarking, can solve the Hénon–Heiles system in a few seconds of wall-clock time; this is compared to the machine learning modelling which can take around one hour of wall-clock time), we have demonstrated that neural initial value problems are a complementary approach which may be used to verify the validity of other numerical methods, for example in highly chaotic regimes. Whilst not presented in the current work, we have also found that coupled neural networks — alongside the neural initial value problem framework — can solve complex-valued initial value problems by treating the underlying differential equations as a system of ordinary differential equations for the

real and imaginary parts (i.e., considering the complex domain to be an ordered pair of real numbers). The methods presented in this chapter could be augmented with the presence of training data (see a general discussion of combining observed data, differential equations, and machine learning in, for example, [Cuo+22]), where the probabilistic activation functions could provide greater accuracy and/or generalisation capabilities for a broader range of dynamical systems.

5

NEURAL STURM–LIOUVILLE PROBLEM

Sturm–Liouville problems are ubiquitous in many areas of physics. The time-independent Schrödinger equation $H\phi(x) = E\phi(x)$, the Laplace equation $\nabla^2\phi(x) = 0$ (and the related Helmholtz equation $\nabla^2\phi(x) = -k^2\phi(x)$), Bessel's equation, and the (generalised) Legendre equation may all be written as a Sturm–Liouville problem. They are all are eigenvalue problems with a unique correspondence between a particular eigenfunction and eigenvalue (i.e., in principle there are no degenerate solutions). We introduce a machine learning method which reliably and accurately finds eigenfunctions and eigenvalues of a broad range of Sturm–Liouville problems through a dual optimisation scheme. An initial guess of

the eigenvalue is input into a machine learning algorithm. The nearest acceptable eigenvalue and its associated eigenfunction are simultaneously found using the neural initial value problem framework. We observe that eigenvectors are found first, followed by the corresponding eigenvalues. Unlike other numerical methods, the eigenvalues can be found from above or below. We use the *neural Sturm–Liouville problem* framework to solve the (generalised) Legendre equation (and then finding the spherical harmonics with post-processing) and the time-independent Schrödinger equation (in both a harmonic and anharmonic trap)—this framework is a fast and reliable *replacement* of traditional numerical schemes (such as the shooting method in appendix C) to solve such problems.

5.1 Statement of the physical problem

We consider the Sturm–Liouville problem

$$\mathcal{L}[\phi(x)] = \lambda w(x)\phi(x), \tag{5.1}$$

where $\mathcal{L}$ is a linear operator, $\phi(x)$ is an eigenfunction, λ is its associated eigenvalue, and $w(x) : \mathbb{R} \to \mathbb{R}$ is a weight function. The domain is typically $x \in [a, b]$, or where appropriate, $x \in (-\infty, \infty)$. The operator $\mathcal{L}$ takes the form

$$\mathcal{L} = -\frac{\mathrm{d}}{\mathrm{d}x}\left[p(x)\frac{\mathrm{d}}{\mathrm{d}x}\right] + q(x), \tag{5.2}$$

where $p(x)$, $p'(x)$, and $q(x)$ are continuous real-valued functions on $[a, b]$. This leads to the standard Sturm–Liouville equation:

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left[p(x)\frac{\mathrm{d}\phi}{\mathrm{d}x}\right] + q(x)\phi = \lambda w(x)\phi. \tag{5.3}$$

We focus on problems with Dirichlet boundary conditions: $\phi(a) = \alpha$ and $\phi(b) = \beta$. The Sturm–Liouville problem has several key properties:

- it possesses infinitely many eigenvalues, each corresponding uniquely to an eigenfunction;
- the eigenvalues are positive and form an increasing sequence: $\lambda_0 < \lambda_1 < \lambda_2 < \cdots < \lambda_n < \cdots \to \infty$ (there always exists a *smallest* (possibly non-zero) eigenvalue λ_0);
- the eigenfunction $\phi_k(x)$ has exactly k zeros in the interval $[a, b]$ and
- the eigenfunctions form an orthonormal basis with respect to the weight function $w(x)$, satisfying

$$\langle \phi_i | \phi_j \rangle = \int_a^b \mathrm{d}x \, \phi_i(x) \phi_j(x) w(x) = \delta_{ij}, \tag{5.4}$$

  where δ_{ij} is the Kronecker delta.

All second-order linear homogeneous ordinary differential equations can be transformed into a Sturm–Liouville equation using an appropriate integrating factor. The time-independent Schrödinger equation, and equations describing phenomena modelled by the Laplace, Helmholtz, Bessel, or Legendre equations, may all be written in Sturm–Liouville form.

5.2 Statement of the machine learning problem

5.2.1 Numerical considerations and cost function

We will denote the neural network's representation of the solution by $\boldsymbol{\phi}(\mathbf{x}) = (\phi_0, \phi_1, \cdots, \phi_n, \cdots, \phi_{N_x})$. Each element of the neural representation, ϕ_n, is associated with the continuous solution at a particular spatial position $x_n = n\Delta x$, i.e., $\phi_n \approx \phi(n\Delta x)$, where $\Delta x = (b - a)/N_x$ is the step size. We are seeking discretised vector representations $\{\boldsymbol{\phi}(\mathbf{x})\}$ of the continuous functions $\{\phi_i(x)\}$ over the spatial domain $\mathbf{x} = (x_0, x_1, \cdots, x_n, \cdots, x_{N_x})$.

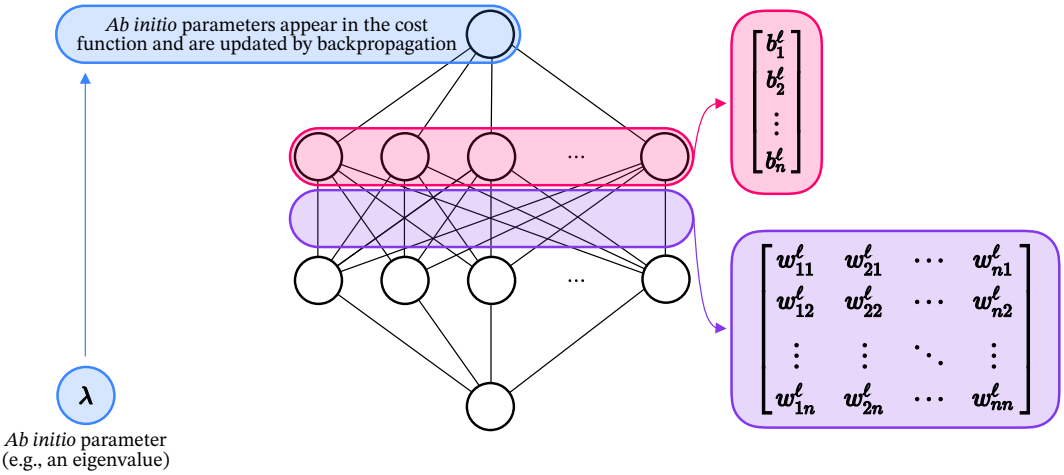


Figure 5.1: The neural network representation of a neural Sturm–Liouville problem. Compare with Figure 4.7: an eigenvalue, λ, is introduced as a global parameter which is not part of the usual structure of the neural network; it appears only in the cost function and is updated using e.g., Adam.

In contrast to the neural initial value problem, we believe that it is in general a requirement that the normalisation condition is satisfied throughout learning to avoid ambiguity in the magnitude of eigenfunctions and to regularise the training process. We introduce the normalisation constraint explicitly in the cost function. The discretised inner product used in the orthonormality condition in equation (5.4) is

$$\langle \phi_i | \phi_j \rangle = \sum_{k=0}^{N_x} \phi_i(x_k)\phi_j(x_k)w(x_k)\Delta x \approx \delta_{ij}. \tag{5.5}$$

The cost function for the nth eigenstate is therefore

$$\begin{aligned} \mathcal{C} = \|\mathcal{L}[\boldsymbol{\phi}(\mathbf{x})] - \lambda \mathbf{w}(\mathbf{x})\boldsymbol{\phi}(\mathbf{x})\|^2 + \left[\Delta x \sum_{i=0}^{N_x} \phi_i^2 w(x_i) - 1\right]^2 \\ + (\phi_0 - \alpha)^2 + (\phi_{N_x} - \beta)^2. \end{aligned} \tag{5.6}$$

The eigenvalue, λ, is a global parameter in the neural network, distinct from weights and biases. It appears explicitly in the cost function but not in the network's

main structure, as shown in Figure 5.1. As a global parameter, λ is not associated with any specific layer or neuron, leaving equations (4.33–4.35) unchanged.

The eigenvalue which is to be learnt adds an extra dimension to the cost landscape (see § 2.5.1 for the dimensional interpretation of neural network parameters). We initialise the eigenvalue, just as we do with weights and biases, but instead we provide a best guess at its value (unlike with the initialisation of the weights and biases which have no physical interpretation). Its value is updated through optimisation algorithms like gradient descent:

$$\lambda_{\mathcal{E}+1} = \lambda_{\mathcal{E}} - \eta \, \Delta\lambda_{\mathcal{E}}, \tag{5.7}$$

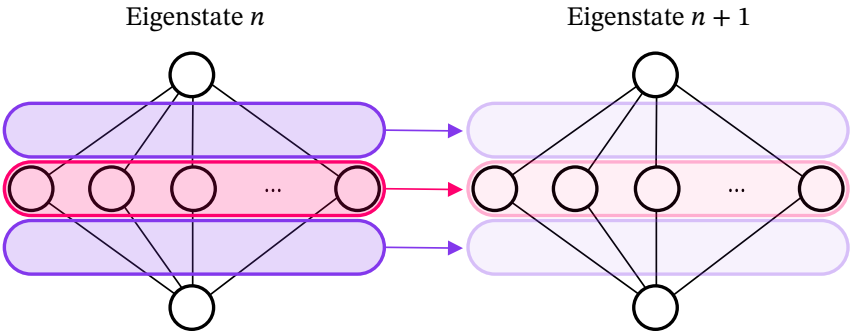
where $\lambda_{\mathcal{E}}$ is λ's value at epoch $\mathcal{E}$. For all examples in this chapter, we use Adam (see a discussion in § 2.5.3) to optimise λ along with other parameters.

5.2.2 Constraining orthogonality with curriculum learning

It is possible—although strictly not a requirement of the neural Sturm–Liouville problem—to constrain eigenstates to be orthogonal. This requires a form of *curriculum learning* [Ben12].[1] We accumulate previously learnt eigenfunctions in memory, determine the weighted overlap $\langle \phi_i | \phi_j \rangle$ for distinct eigenstates $i \neq j$, and add this overlap to the cost function as an additional regularisation term. We start by learning the lowest eigenstate first, although in principle this is not a requirement. Curriculum learning also allows us to use the learnt weights and biases for the first eigenstate as a foundation for the second eigenstate, and so on. This approach would be beneficial for learning many eigenstates at a time.

Figure 5.2 shows a schematic of curriculum learning within the neural Sturm–Liouville problem.

[1] "Humans and animals learn much better when the examples are not randomly presented but organized in a meaningful order which illustrates gradually more concepts, and gradually more complex ones. [Curriculum learning formalises] such training strategies in the context of machine learning [...]." – Bengio, *et al.*, [Ben12, op cit.]



Curriculum learning: the optimised weights and biases for the nth eigenstate are used as the initial state for the $n+1$ eigenstate.

Figure 5.2: A schematic for curriculum learning with shared weights and biases. The weights (in between layers, in colour: purple) and biases (associated with each neuron, in colour: pink) from the optimised nth eigenstate form the initial state for the $n+1$th eigenstate. Orthogonality between each state is also satisfied in the cost function.

We introduce the full orthonormality constraint—in turn introducing curriculum learning for the eigenfunctions—explicitly in the cost function. The cost function for the nth eigenstate is

$$\begin{aligned}\mathcal{C} = \|\mathcal{L}[\boldsymbol{\phi}(\mathbf{x})] - \lambda \mathbf{w}(\mathbf{x})\boldsymbol{\phi}(\mathbf{x})\|^2 + \sum_{k=0}^{n}\left[\Delta x \sum_{i=0}^{N_x} \phi_i^j \phi_i^n w(x_i) - \delta_{jn}\right]^2 \\ + (\phi_0 - \alpha)^2 + (\phi_{N_x} - \beta)^2,\end{aligned} \tag{5.8}$$

where, in the second term, the superscript refers to a particular eigenstate and the subscript refers to the spatial index of that eigenstate. We again assume that the cost function is dimensionless.

5.3 Solving the Legendre equation and generating the spherical harmonics

5.3.1 Statement of the problem and cost function

The Legendre equation and its solutions are ubiquitous in physics: in quantum mechanics, they play a crucial role in the solutions of the Schrödinger equation

for systems with spherical symmetry, such as the hydrogen atom. The angular part of the wave function is expressed in terms of spherical harmonics, which are constructed using associated Legendre polynomials. These functions describe the angular dependence of the electron's probability distribution. Furthermore, the quantisation of angular momentum in quantum mechanics is intimately connected with the properties of spherical harmonics, providing discrete eigenvalues for the angular momentum operators. This quantisation is fundamental to understanding atomic and molecular spectra, electron orbitals, and the selection rules for quantum transitions. The solutions are analytically known, and we use the Legendre equation as a test problem for our framework.

The general Legendre equation is written in Sturm–Liouville form as

$$\frac{\mathrm{d}}{\mathrm{d}x}\left[(1-x^2)\frac{\mathrm{d}}{\mathrm{d}x}P_\ell^m(x)\right]+\left[\ell(\ell+1)-\frac{m^2}{1-x^2}\right]P_\ell^m(x)=0, \tag{5.9}$$

where the eigenfunctions are $P_\ell^m(x)$. We choose $\ell(\ell+1)$ as the eigenvalue which ensures that the weighting function $w(x)$ in the definition of the Sturm–Liouville theory is unity.[2] The eigenfunctions are called the *associated Legendre polynomials*. The domain of the problem is $x \in [-1,1]$. We choose a boundary condition $P_\ell^0(1)=1$ to standardise all eigenfunctions.[3] The eigenfunctions obey the completeness and orthogonality properties of the Sturm–Liouville theory, i.e.,

$$\int_{-1}^{1}\mathrm{d}x\,P_k^m(x)P_\ell^m(x)=\frac{2(\ell+m)!}{(2\ell+1)(\ell-m)!}\delta_{k\ell}, \tag{5.10}$$

where $\delta_{k\ell}$ is the Kronecker delta. Ensuring single-valuedness and avoiding issues with infinities in physical problems, ℓ is restricted to non-negative integer values and m is allowed only the $2\ell+1$ values $-\ell,-(\ell+1),\cdots,-1,0,+1,\cdots,\ell-1,\ell$.

[2] If m^2 is chosen as the eigenvalue, then the weighting function is $w(x)=(1-x^2)^{-1}$.

[3] We prefer to use the term 'standardisation' rather than 'normalisation' to constrain the norm of the eigenfunctions, since the norm is not equal to unity.

In order to avoid issues with infinities in equation (5.9) with the division of $1-x^2$, we shall introduce the change of variable $P_\ell^m(x) = (1-x^2)^{m/2}Q_\ell^m(x)$. This leads to a form of the general Legendre equation which is compatible with numerical studies,

$$(1-x)^2\frac{\mathrm{d}^2}{\mathrm{d}x^2}Q_\ell^m(x) - 2(m+1)x\frac{\mathrm{d}}{\mathrm{d}x}Q_\ell^m(x) + [\ell(\ell+1) - m(m+1)]Q_\ell^m(x) = 0. \quad (5.11)$$

The cost function we wish to minimise is

$$\begin{aligned}\mathcal{C} = &\left\|(\mathbb{1}-\mathbf{x})^2\frac{\mathrm{d}^2}{\mathrm{d}\mathbf{x}^2}\mathbf{Q}_\ell^m(\mathbf{x}) - 2(m+1)\mathbf{x}\frac{\mathrm{d}}{\mathrm{d}\mathbf{x}}\mathbf{Q}_\ell^m(\mathbf{x}) + [\ell(\ell+1) - m(m+1)]\mathbf{Q}_\ell^m(\mathbf{x})\right\|^2 \\ &+ \left[\Delta x\sum_{i=0}^{N_x}\|(Q_\ell^m)_i\|^2 - \frac{2(\ell+m)!}{(2\ell+1)(\ell-m)}\right]^2 + \left[(Q_\ell^m)_0 - s_0\right]^2 + \left[(Q_\ell^m)_{N_x} - s_{N_x}\right]^2,\end{aligned} \quad (5.12)$$

where $s_{N_x} = 1$ if $m = 0$ and $s_{N_x} = 0$ if $m \neq 0$ and $\left[(Q_\ell^m)_0 - s_0\right]^2$ is only necessary if $m \neq 0$, where $s_0 = 0$. The four terms respectively minimise the differential equation, a normalisation condition, the leftmost boundary condition (for $m \neq 0$) and the rightmost boundary condition (for all m).

Given knowledge of the Legendre polynomials, one can find another set of orthogonal functions known as the *spherical harmonics*, $Y_\ell^m(\theta,\varphi)$, which are the eigenfunctions of angular momentum,

$$\boldsymbol{\ell}^2 Y_\ell^m(\theta,\varphi) = \lambda Y_\ell^m(\theta,\varphi), \quad (5.13)$$

where the operator $\boldsymbol{\ell}^2$ represents the square of the total angular momentum, while its components ℓ_x, ℓ_y, and ℓ_z correspond to angular momentum along the respective axes. We note that the *components* of angular momentum do not commute with each other, i.e., $[\ell_x,\ell_y] = i\hbar\ell_z$, $\quad[\ell_y,\ell_z] = i\hbar\ell_x$, $\quad[\ell_z,\ell_x] = i\hbar\ell_y$, but the square of the angular momentum operator does compute with each component, $[\boldsymbol{\ell}^2,\ell_x] = [\boldsymbol{\ell}^2,\ell_y] = [\boldsymbol{\ell}^2,\ell_z] = 0$. This allows us to find simultaneous eigenfunctions of $\boldsymbol{\ell}^2$ and

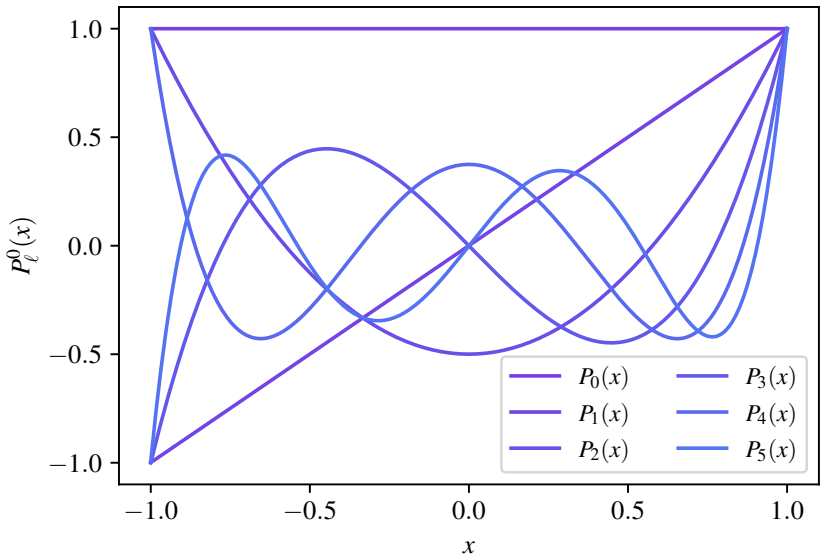


Figure 5.3: The first six Legendre polynomials, P_ℓ^m, with $m = 0$, produced by the neural Sturm–Liouville problem. We will not use all of these polynomials in our calculations of the spherical harmonics.

one component, conventionally chosen as ℓ_z. These eigenfunctions are the spherical harmonics where

$$\boldsymbol{\ell}^2 Y_\ell^m = \ell(\ell+1)\hbar^2 Y_\ell^m, \tag{5.14}$$

and

$$\ell_z Y_\ell^m = m\hbar Y_\ell^m. \tag{5.15}$$

The spherical harmonics may be defined in terms of the associated Legendre functions

$$Y_\ell^m(\theta, \varphi) = P_\ell^m(\cos\theta)\mathrm{e}^{im\varphi}. \tag{5.16}$$

5.3.2 Results and discussion

We want to visualise the spherical harmonics up to and including order $m = 3$. We achieve this by post-processing the associated Legendre functions which are obtained by the neural Sturm–Liouville problem. We first find the Legendre polynomials of

order $m = 0$. We can either use the neural Sturm–Liouville problem again for the remaining orders, or recognise that the higher-order functions may be determined more efficiently providing we have the $m = 0$ polynomials:

$$P_\ell^m(x) = (-1)^m(1-x^2)^{m/2}\frac{\mathrm{d}^m}{\mathrm{d}x^m}P_\ell^0(x), \tag{5.17}$$

where we include the Condon–Shortley phase factor $(-1)^m$ and compute the derivatives using a numerical method such as autodifferentiation.

Figure 5.3 shows the first six Legendre polynomials of order $m = 0$ produced by machine learning methods. To ascertain the error of the eigenstates from the neural Sturm–Liouville problem, we determine the overlap between these machine learning solutions and the analytic Legendre polynomials, $\langle \mathrm{ML}|\mathrm{analytic}\rangle$. We determine the overlap using Simpson's integration rule (so any value of the overlap is only as good as the error in Simpon's rule). Figure 5.4 shows the numerical overlap for $m = 0$.

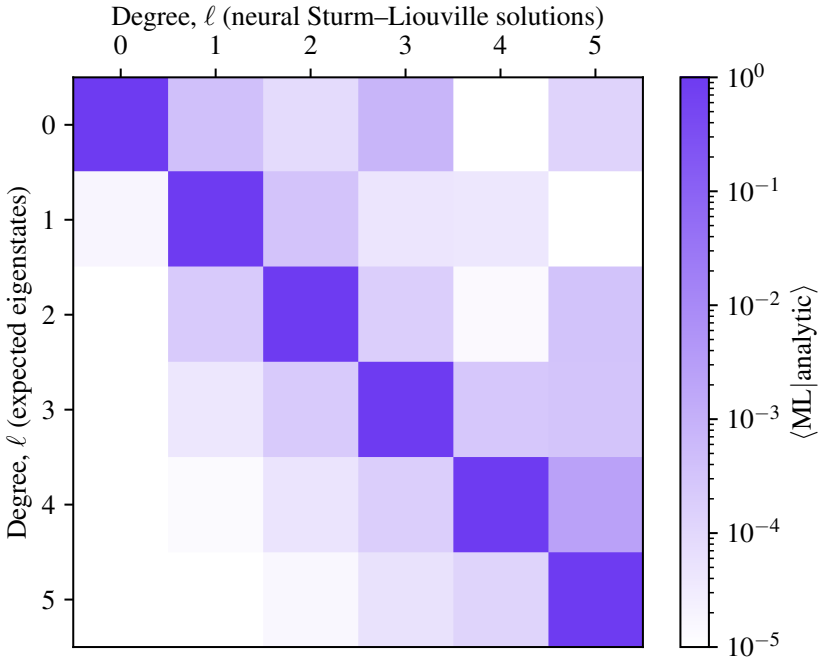


Figure 5.4: The overlap $\langle \mathrm{ML}|\mathrm{analytic}\rangle$ of the eigenstates of the Legendre equation produced by machine learning and the analytic results. The colour map assumes a logarithmic scale.

Figure 5.5 shows the individual eigenstates $n = 0$ to $n = 5$ alongside the estimated eigenvalue, λ, and the root-mean-square of the ODE. The eigenvalues may increase, decrease, or both, during the training. The eigenvalues may overshoot their expected value during the training, possibly associated with a hallucination of a different solution to the Legendre equation.

Figure 5.6 shows the magnitude $|Y_\ell^m(x)|^2$ of the spherical harmonics. Figure 5.7 shows the real part of the spherical harmonics. Figure 5.8 shows the imaginary part of the spherical harmonics—note that $m = 0$ harmonics have no imaginary component.

The spherical harmonics $Y_l^m(\theta, \varphi)$ are determined by

$$Y_l^m(\theta, \varphi) = (-1)^m N_l^m P_l^{|m|}(\cos\theta) e^{im\varphi} \tag{5.18}$$

where N_l^m is the normalisation factor given by

$$N_l^m = \sqrt{\frac{(2l+1)}{4\pi}\frac{(l-|m|)!}{(l+|m|)!}}, \tag{5.19}$$

which is known *a priori* from Sturm–Liouville theory. For computational efficiency and to handle the case of negative m values, we implement this as follows:

$$Y_l^m(\theta, \varphi) = \begin{cases} N_l^m P_l^m(\cos\theta)(\cos(m\varphi) + i\sin(m\varphi)) & \text{if } m > 0 \\ N_l^{|m|} P_l^{|m|}(\cos\theta) & \text{if } m = 0 \\ N_l^{|m|} P_l^{|m|}(\cos\theta)(\sin(|m|\varphi) - i\cos(|m|\varphi)) & \text{if } m < 0 \end{cases} \tag{5.20}$$

This formulation ensures that the spherical harmonics satisfy the conventional condition $Y_l^{-m} = (-1)^m (Y_l^m)^*$, where * denotes complex conjugation.

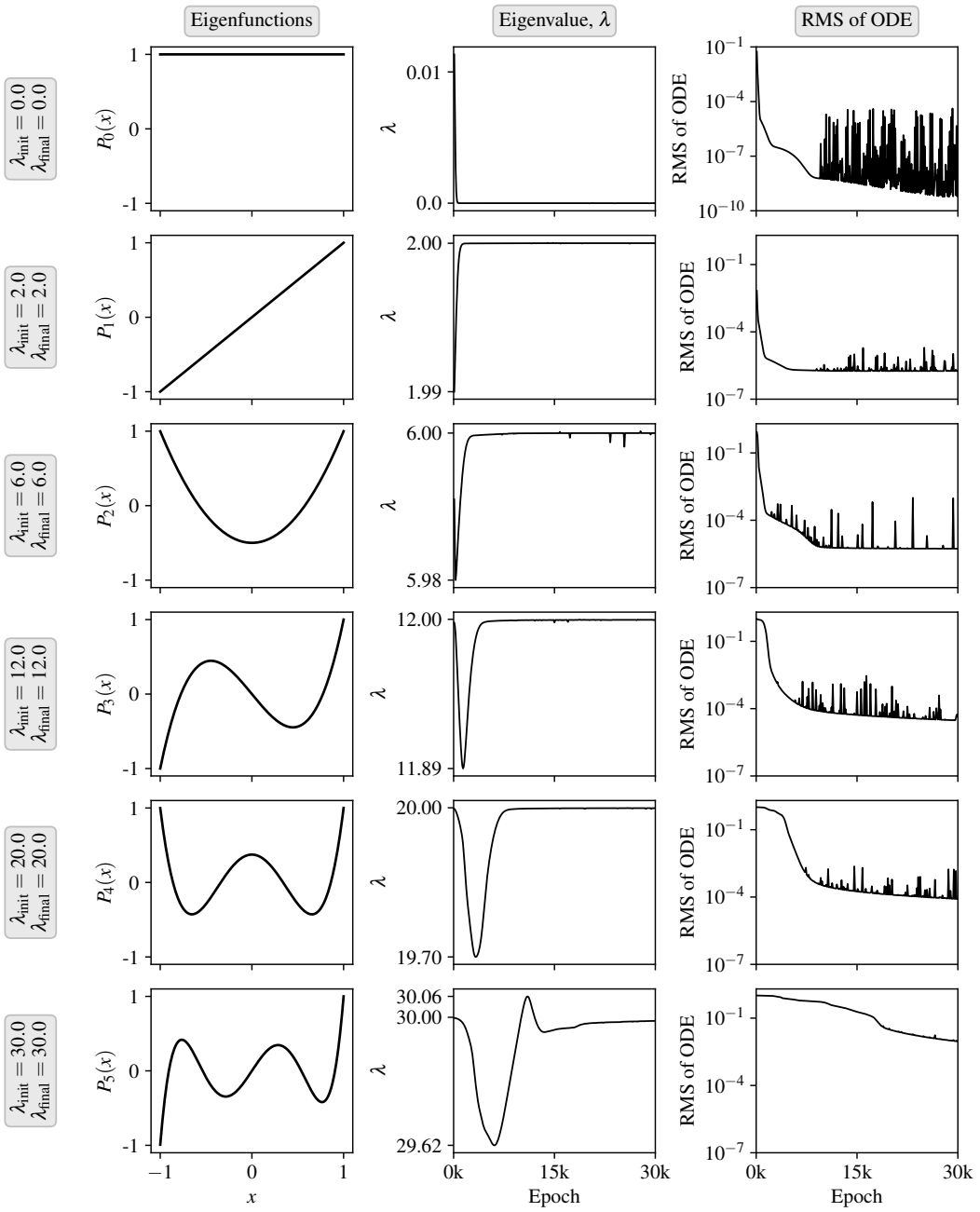


Figure 5.5: Legendre polynomials and training metrics for $l = 0$ to $l = 5$. Left column: Neural network approximations of $P_\ell^0(x)$. Middle column: Evolution of eigenvalues λ during training. We label the maximum (where appropriate), minimum and expected values of the eigenvalues. Right column: the root-mean-square of the ODE as a function of the number of epochs on a logarithmic scale

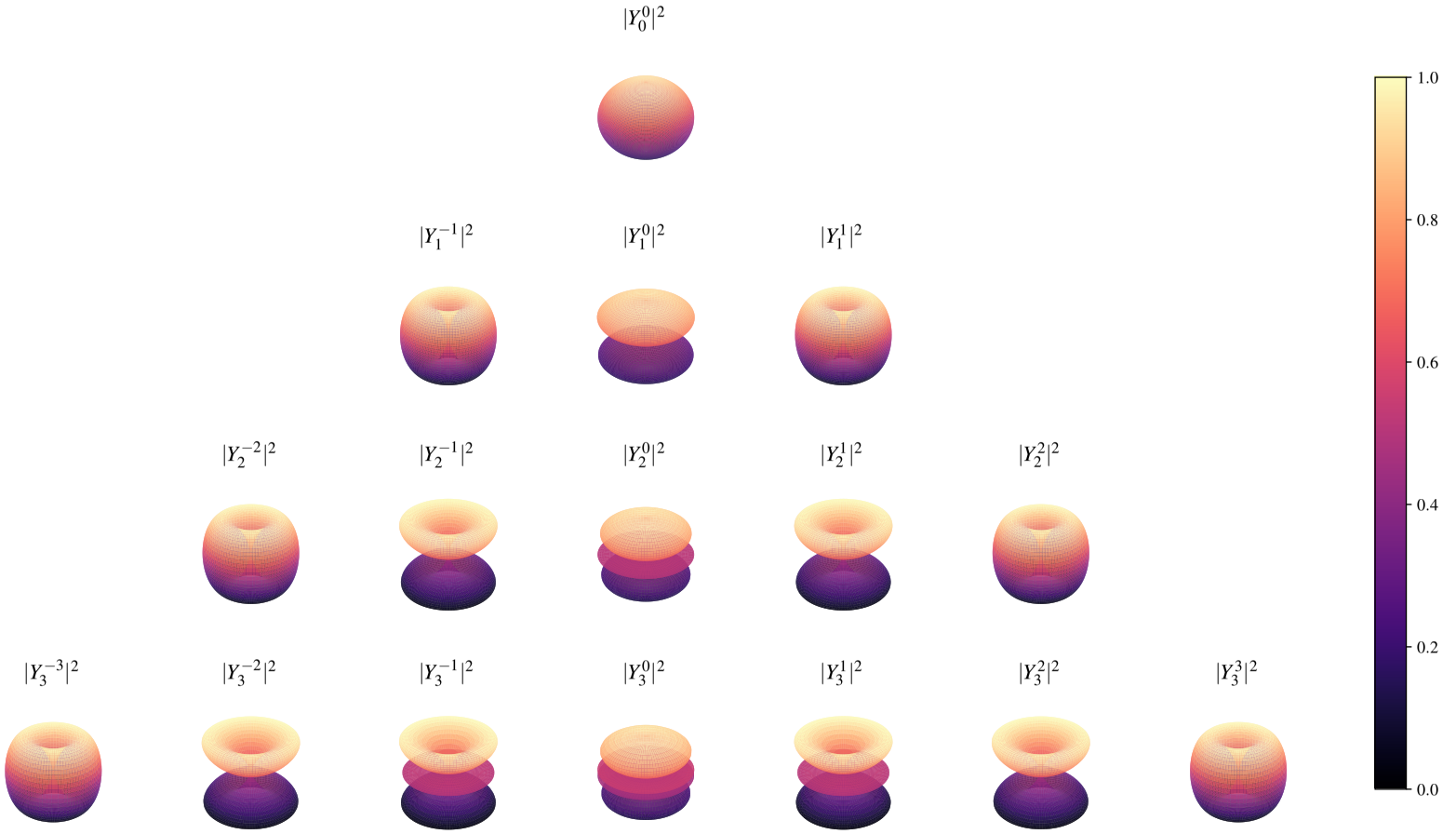


Figure 5.6: Visualisation of the magnitude, $|Y_l^m(\theta, \varphi)|^2$, of the spherical harmonics for $l = 0, 1, 2, 3$ and $-l \leq m \leq l$. All spherical harmonics are produced by post-processing the machine learnt associated Legendre polynomials. Each subplot represents a specific (l, m) combination, with the radial distance from the origin proportional to the magnitude squared of the spherical harmonic. The colour gradient indicates the probability density, with brighter colours representing higher values. This figure illustrates the probability distribution of finding a particle in different angular positions for various quantum states described by spherical harmonics.

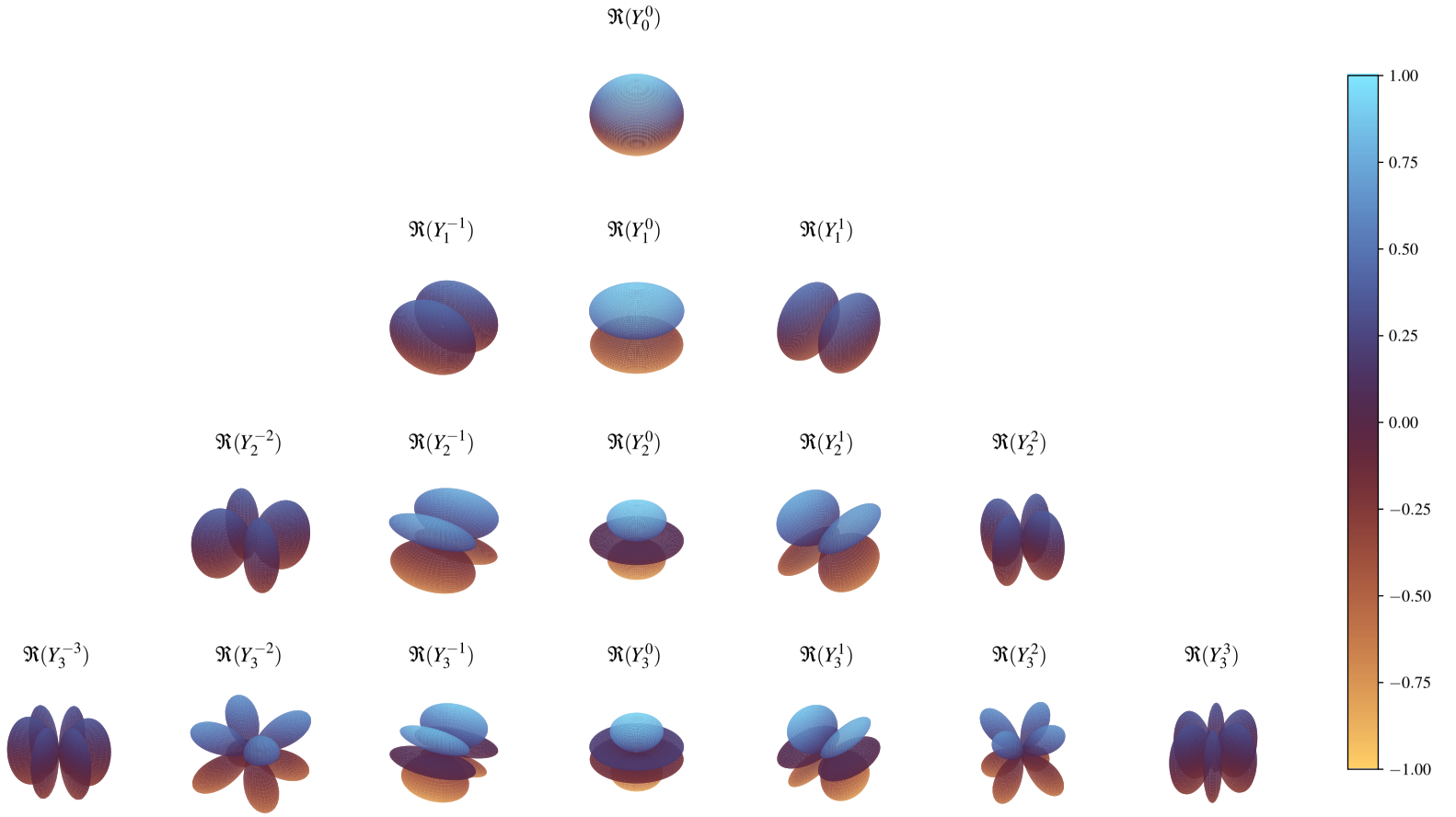


Figure 5.7: Three-dimensional visualisation of the real part, $\Re(Y_l^m(\theta, \varphi))$, of the spherical harmonics for $l = 0, 1, 2, 3$ and $-l \leq m \leq l$. All spherical harmonics are produced by post-processing the machine learnt associated Legendre polynomials. Each subplot shows the real component of a specific (l, m) spherical harmonic, with the radial distance from the origin proportional to the absolute value of the real part.

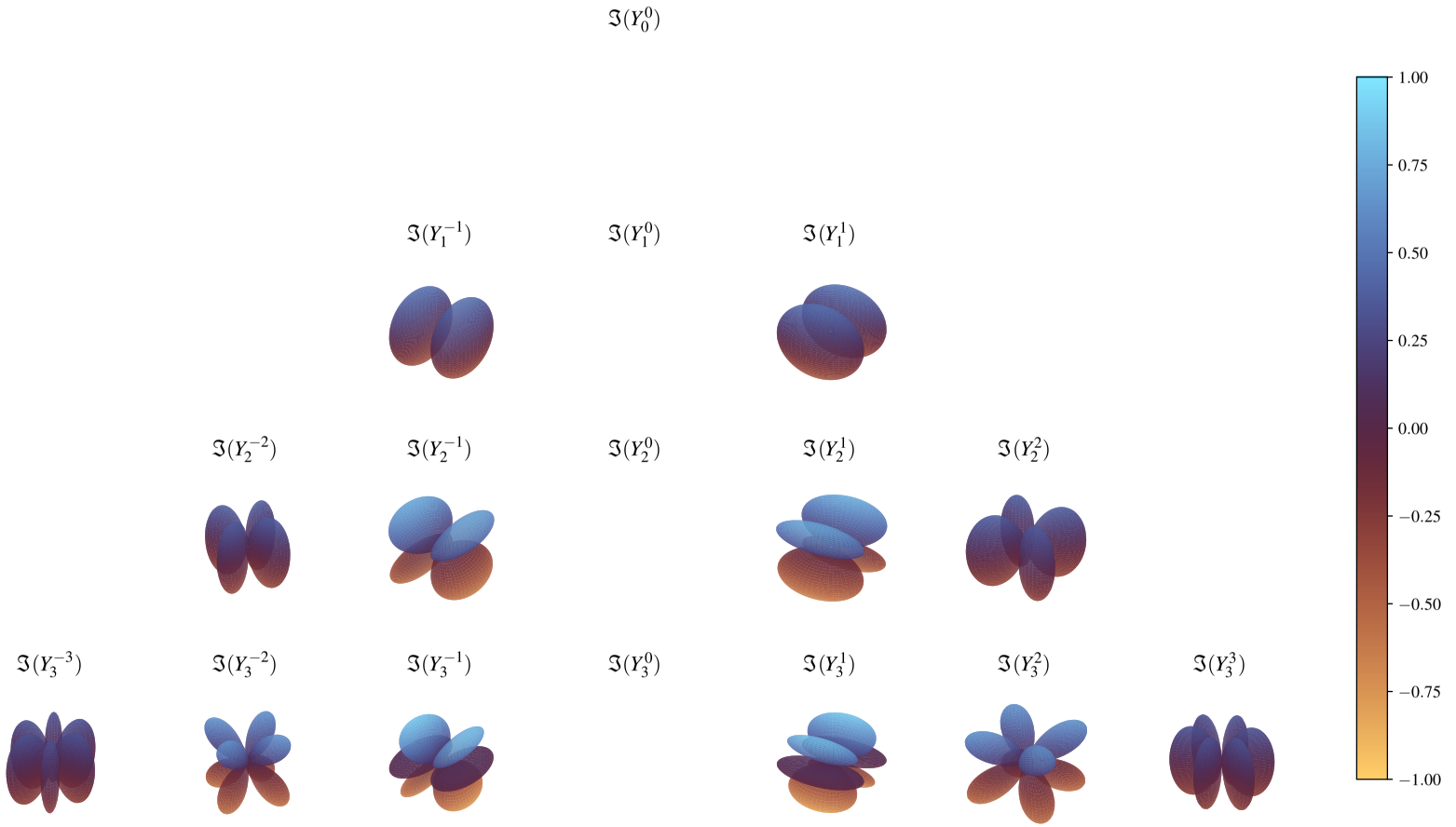


Figure 5.8: Three-dimensional visualisation of the imaginary part, $\Im(Y_l^m(\theta, \varphi))$, of the spherical harmonics for $l = 0, 1, 2, 3$ and $-l \leq m \leq l$. All spherical harmonics are produced by post-processing the machine learnt associated Legendre polynomials. Each subplot represents the imaginary component of a specific (l, m) spherical harmonic, with the radial distance from the origin proportional to the absolute value of the imaginary part. There is no imaginary component associated with the $m = 0$ spherical harmonics.

5.4 Solving the time-independent Schrödinger equation

5.4.1 Harmonic trap

Consider the time-independent Schrödinger equation in one dimension in a harmonic trap

$$-\frac{\hbar^2}{2m}\frac{\mathrm{d}^2\phi_n(x)}{\mathrm{d}x^2} + \frac{1}{2}m\omega^2x^2\phi_n(x) = E_n\phi_n(x), \tag{5.21}$$

where $n \geq 0$ is the quantum number, m is the mass of the particle in the trap, ω is the angular frequency of the trap and E_n is the energy eigenvalue. To ensure a dimensionless form, we scale the position, x, by the harmonic oscillator length to obtain the dimensionless position

$$\xi = \sqrt{\frac{m\omega}{\hbar}}x. \tag{5.22}$$

We now associate $\phi_n(\xi)$ as the dimensionless eigenstates of the quantum harmonic oscillator. We scale the corresponding eigenenergies by the characteristic energy $\hbar\omega$ to obtain the dimensionless energy

$$\varepsilon_n = \frac{E_n}{\hbar\omega}. \tag{5.23}$$

Equations (5.22) and (5.23) lead to, after some rearrangement, the dimensionless time-independent Schrödinger equation

$$\frac{\mathrm{d}^2\phi_n(\xi)}{\mathrm{d}\xi^2} = \left(\frac{1}{2}\xi^2 - \varepsilon_n\right)\phi_n(\xi). \tag{5.24}$$

We are seeking normalisable eigenstates, $\phi_n(\xi)$, and their corresponding energies, ε_n, subject to the boundary conditions

$$\phi_n(\xi \to -\infty) = \phi_n(\xi \to +\infty) = 0. \tag{5.25}$$

This problem has well-known normalised solutions (obtained through either ladder algebra or asymptotic methods)

$$\phi_n(\xi) = \frac{1}{\sqrt{2^n n! \sqrt{\pi}}} \mathrm{e}^{-\xi^2/2} H_n(\xi), \tag{5.26}$$

where $H_n(\xi)$ are the physicist's Hermite polynomials—a set of orthogonal polynomials defined by the formula

$$H_n(\xi) = (-1)^n \mathrm{e}^{\xi^2} \frac{\mathrm{d}^n}{\mathrm{d}\xi^n} \left(\mathrm{e}^{-\xi^2}\right). \tag{5.27}$$

The corresponding dimensionless energy eigenvalues ε_n are given by $\varepsilon_n = n + \frac{1}{2}$.

The cost function we wish to minimise is

$$\mathcal{C} = \left\| \frac{1}{2} \frac{\mathrm{d}}{\mathrm{d}\mathbf{x}} \boldsymbol{\phi}(\mathrm{x}) + (V - E)\boldsymbol{\phi}(\mathbf{x}) \right\|^2 + \left(\Delta x \sum_{i=0}^{N_x} \|\phi_i\|^2 - 1 \right)^2 + (\phi_0 - s_0)^2 + \left(\phi_{N_x} - s_{N_x}\right)^2, \tag{5.28}$$

where in all cases $s_0 = s_{N_x} = 0$ to satisfy the boundary conditions in equation (5.25). The four terms respectively minimise the differential equation, a normalisation condition, the leftmost boundary condition and the rightmost boundary condition.

5.4.2 Anharmonic trap

We now consider a time-independent perturbation that introduces anharmonicity to the potential. Working again in a dimensionless unit system, the perturbation to the Hamiltonian is

$$H' = \alpha \xi^4, \tag{5.29}$$

where α is a small, dimensionless parameter characterising the strength of the anharmonicity. We will now determine through perturbation theory the shifted energies as a consequence of this perturbation—this does not appear in the neural Sturm–Liouville problem and is used only to check the validity of our solutions. We

consider the first-order correction to the dimensionless energy which is given by the expectation value of the perturbation in the unperturbed eigenstates

$$\Delta\varepsilon_n^{(1)} = \langle\phi_n|H'|\phi_n\rangle = \alpha\langle\phi_n|\xi^4|\phi_n\rangle, \tag{5.30}$$

where $\phi_n(\xi)$ are the normalised eigenstates of the unperturbed harmonic oscillator as given in equation (5.26).

The matrix element we need to evaluate is

$$\langle\phi_n|\xi^4|\phi_n\rangle = \int_{-\infty}^{\infty} \phi_n^*(\xi)\xi^4\phi_n(\xi)\mathrm{d}\xi. \tag{5.31}$$

which is a standard Gaussian integral. The matrix element for all n is

$$\langle\phi_n|\xi^4|\phi_n\rangle = \frac{6n^2+6n+3}{4}. \tag{5.32}$$

The first-order correction to the dimensionless energy is thus

$$\Delta\varepsilon_n^{(1)} = \alpha\frac{6n^2+6n+3}{4}. \tag{5.33}$$

The total dimensionless energy up to first order in perturbation theory is therefore

$$\varepsilon_n \approx \varepsilon_n^{(0)} + \Delta\varepsilon_n^{(1)} \tag{5.34}$$

$$= n + \frac{1}{2} + \frac{3\alpha}{4}(2n^2+2n+1). \tag{5.35}$$

The anharmonic perturbation causes an upward shift in the energy levels, with the shift becoming more pronounced for higher energy states. While this first-order correction provides a good approximation for small α and low n, for higher n or larger α, higher-order corrections to the energies may be required for accurate results.

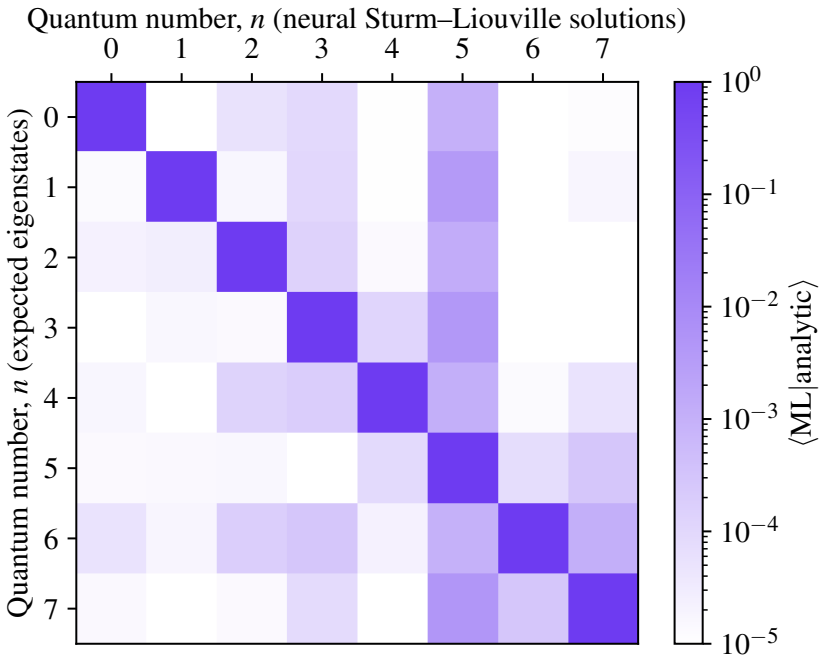


Figure 5.9: The overlap ⟨ML|analytic⟩ of the eigenstates of the quantum harmonic oscillator produced by machine learning and the analytic results. The colour map assumes a logarithmic scale.

5.4.3 Results and discussion

To ascertain the error of the eigenstates from the neural Sturm–Liouville problem, we determine the overlap between these machine learning solutions and the analytic eigenstates of the quantum harmonic oscillator ⟨ML|analytic⟩. We again determine the overlap using Simpson's integration rule. Figure 5.9 shows the numerical overlap for the first eight eigenstates.

Figure 5.10 demonstrates the cost metrics for the eigenstates in a harmonic potential. The neural Sturm–Liouville problem is able to reliably produce the eigenstates sampled. The initial eigenergies were either taken as values above, equal to, or below the expected eigenenergy. We again observe the exploration of unphysical energies during training, which we interpret (akin to § 4.7) as a hallucination of the neural Sturm–Liouville problem.

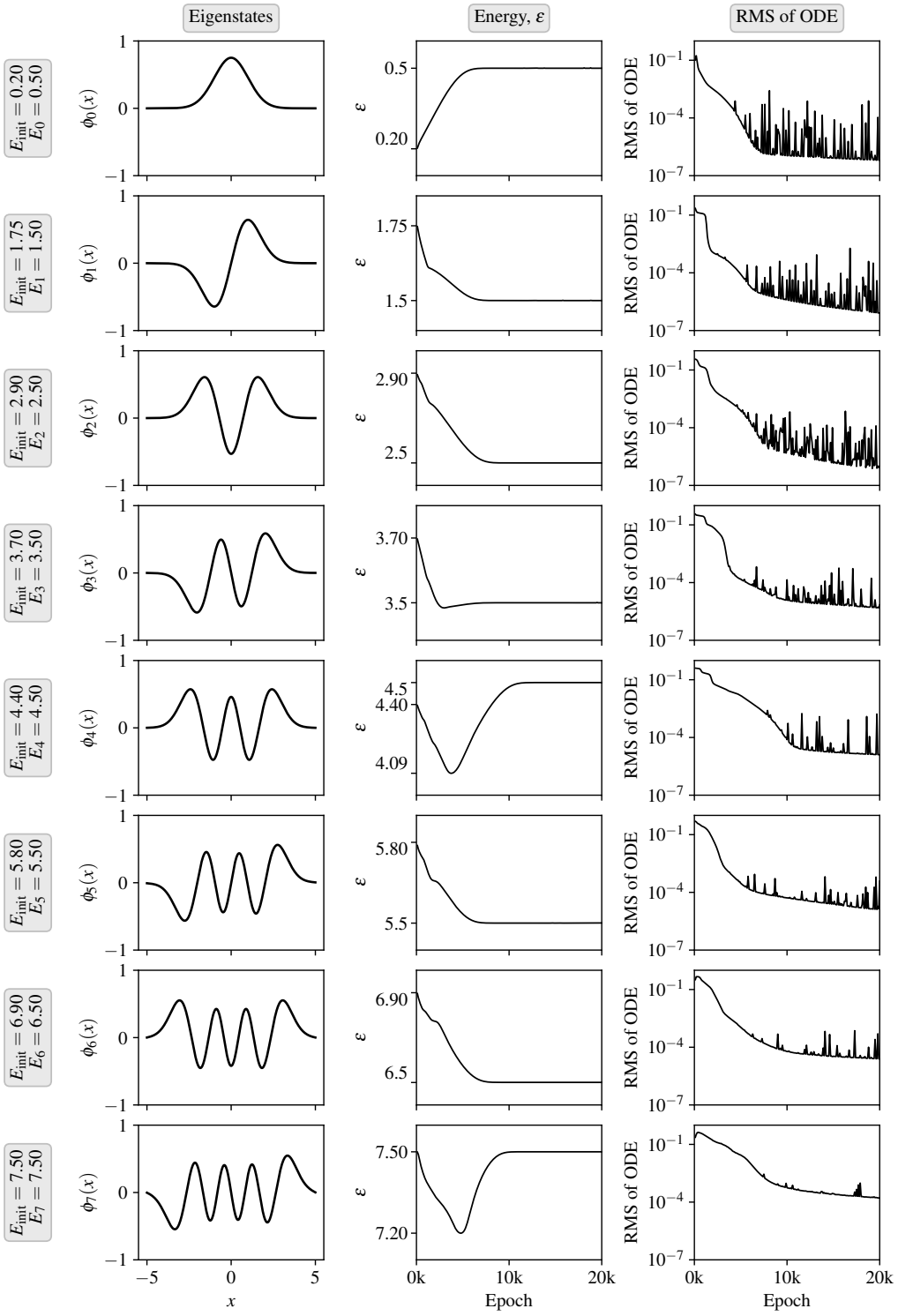


Figure 5.10: Eigenstates with quantum numbers $n = 0$ to $n = 7$. Left column: neural network approximations of each eigenstate. Middle column: evolution of eigenenergies ε during training. We label the maximum (where appropriate), minimum and expected values of the eigenvalues. Right column: the root-mean-square of the ODE as a function of the number of epochs on a logarithmic scale.

We observe that the discretisation of the spatial grid has a role in the reliability of the neural Sturm–Liouville problem—finer grids more reliably and accurately find eigenstates and their corresponding eigenvalues compared to coarse grids.

Figure 5.11 shows the evolution of the eigenstates and eigenvalues during training. We observe that the eigenstates explore all lower-energy eigenstates and that the eigenstate, rather than the eigenvalue, is the first to be optimised.

Figure 5.12 compares the results of the harmonic and anharmonic oscillators. The perturbation in the anharmonic oscillator causes an upwards shift in the energy levels. The percentage difference between the energies produced by the neural Sturm–Liouville problem and the analytical results from first- and second-order perturbation theory range from 0.03% to 0.68%. The neural Sturm–Liouville approach is consistent with what we would expect from a perturbative approach.

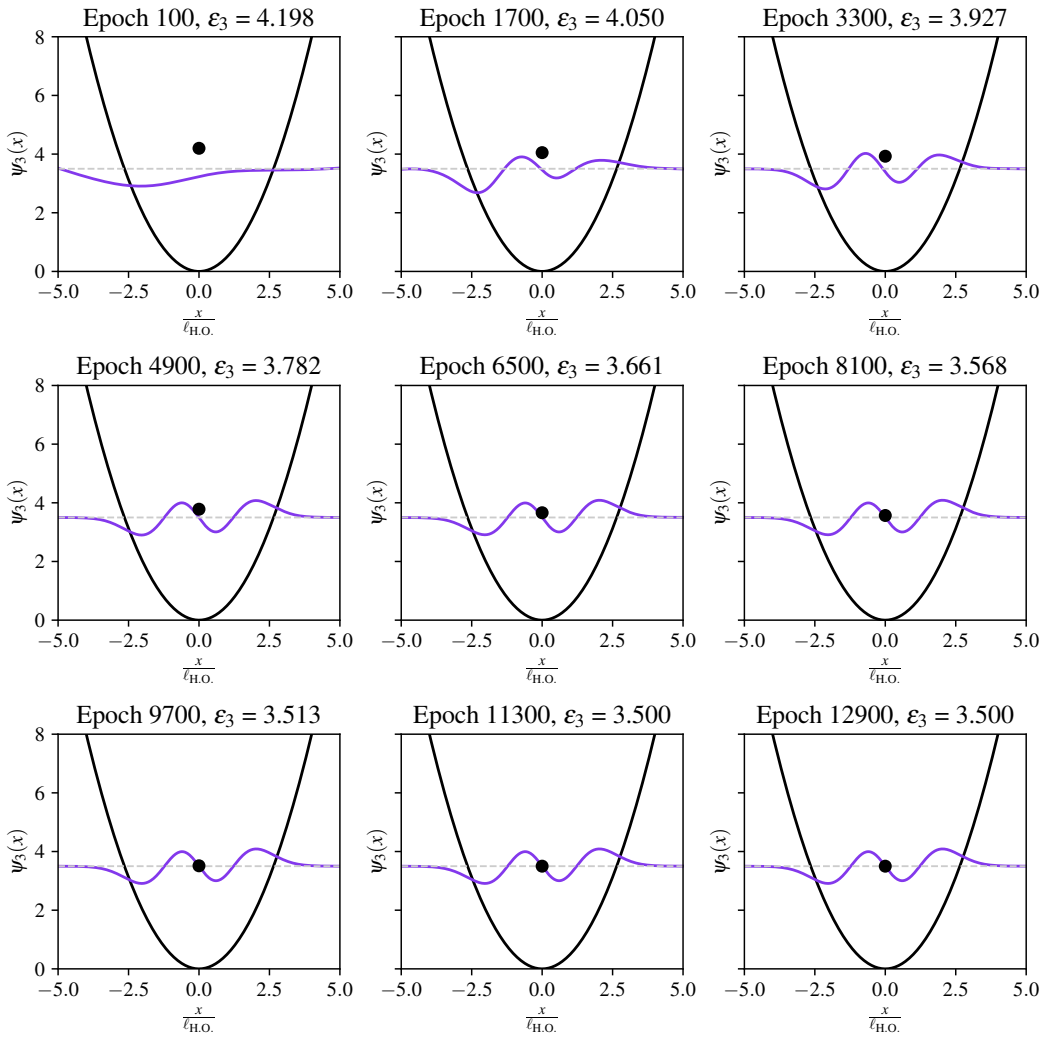


Figure 5.11: The evolution of the $n = 3$ eigenstate (solid purple line) and eigenenergy (black circle) during training. The grey dashed line indicates the expected eigenenergy, $\varepsilon = 7/2$, of that state.

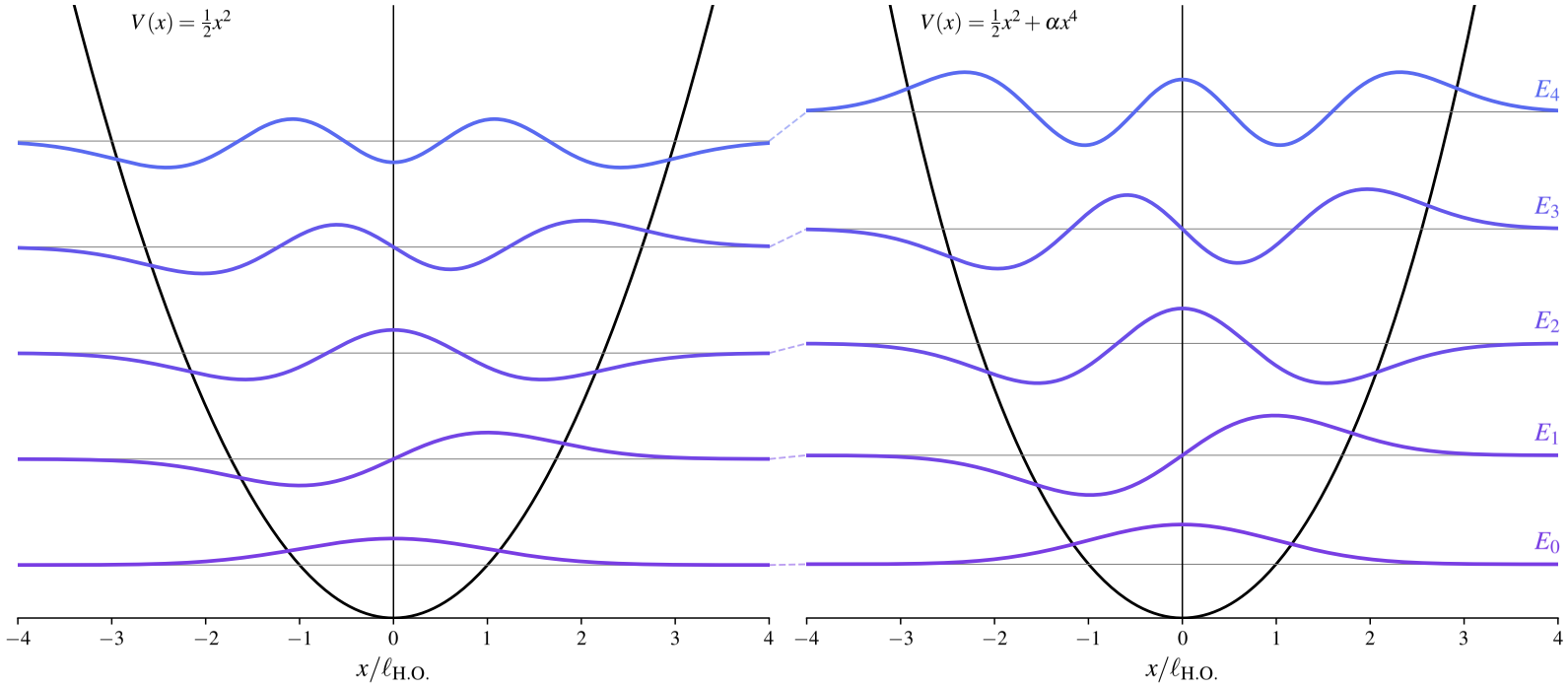


Figure 5.12: The first five bound states of the quantum harmonic oscillator (left), $V(x) = \frac{1}{2}x^2$, and of the quantum anharmonic oscillator (right), $V(x) = \frac{1}{2}x^2 + \alpha x^4$, where $\alpha = 0.01$. The differences between the unperturbed and perturbed energy levels are visible for most eigenstates. The vertical axis corresponds to the energy of that state, ε_n.

5.5 Summary

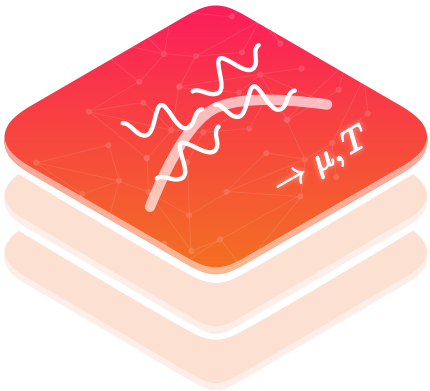
We have presented the neural Sturm–Liouville problem, a novel machine learning method which finds the eigenfunctions and eigenvalues of Sturm–Liouville problems. Demonstrated on the general Legendre equation and the time-independent Schrödinger equation in a harmonic and anharmonic trap, the neural Sturm–Liouville problem is a reliable and quick method to find solutions to a variety of eigenvalue problems which are commonplace in physics. In the examples we have looked at, the neural Sturm–Liouville problem appears to respect the discrete spectra of eigenvalues of these problems.

In our benchmarks, the neural Sturm–Liouville problem takes around 45–60 seconds of wall-clock time, compared to around a second of wall-clock time for traditional eigenvalue problem solvers such as the shooting method. This represents around an order of magnitude difference in wall-clock time.

The neural Sturm–Liouville problem represents an important milestone in our studies of machine learning techniques as they apply to physics-based problems. We have so far demonstrated the ability for machine learning methods to solve a variety of ordinary differential equations, and the extension of the neural initial value problem to eigenvalue problems is a natural one. The neural Sturm–Liouville problem's ability to rapidly produce accurate solutions to such problems positions it as a valuable addition to the physicist's computational toolkit.

6

MACHINE LEARNING THERMODYNAMICAL PARAMETERS OF QUANTUM FLUIDS



BOSE GASES AT THERMAL EQUILIBRIUM are characterised by their chemical potential and temperature. Measuring the temperature and average number of atoms (equivalent to measuring the chemical potential) in either a time-of-flight or *in situ* imaged experimental density profile is often imprecise and destructive. We introduce a proof-of-principle machine learning model which can predict—within fractions of a second—the chemical potential and temperature of

a single-shot density profile of a Bose gas with a thermal component. Whilst only trained on spherically harmonic trapping potentials, we note that—remarkably—the model can predict the chemical potential and temperature of toroidally trapped condensates with reasonable accuracy (around ± a few nanokelvins). The model can also predict these parameters throughout thermalisation after a relatively brief evolution. A machine learning model trained on more geometries, and over a broader range of temperatures and chemical potentials could in principle generate real-time predictions of these parameters (and others including the average atom number and the scattering length) in experiments.

6.1 Description of the model

6.1.1 Training data

We simulate 3,000 Bose-condensed clouds of rubidium-87 atoms with a scattering length $a_s = 5.29\,\text{nm}$ and atomic mass $1.44 \times 10^{-25}\,\text{kg}$ using the stochastic Gross–Pitaevskii equation (see § 3.4 for a description of the physics and numerical implementation). We use a harmonic trap with transverse frequencies $\omega_x = \omega_y = 2\pi \times 25\,\text{Hz}$ and perpendicular frequency $\omega_z = 100\omega_x$. We add some slight anisotropy to the transverse dimensions of the trap of the order of ± few Hz to mimic a possibly imperfect trap or imaging in experiments. We sample the chemical potential from the distribution $\mu \sim \text{Uniform}(20, 80)$ nK and the temperatures from the distribution $T \sim \text{Uniform}(1, 200)$ nK rather than from a regularly sized grid.[1] These ranges encompass both deeply degenerate and near-critical regimes. Other parameters are available in our open source code [Gri24b].

Since we are interested only in producing equilibrium thermal states of the condensate rather than studying the time-dependent process of its formation, the precise value of the dimensionless coupling parameter γ is not critical for our purposes. The

[1] This adds another source of stochasticity to the machine learning model (see § 2.9.2 for a discussion on the importance of stochasticity in machine learning). We observed that the uniform sampling technique improved the predictive capabilities of the network—the predicted values of μ and T had an overall lower absolute error beyond what might be expected from run-to-run variation.

role of γ in our simulations is therefore only to scale the condensation time. We therefore choose a spatially constant rate $\gamma = 0.01$.

We simulate the dynamics of the condensate up to the point when thermal equilibrium is achieved. We determine this by calculating the relative change of the condensate number from the second time step onwards,

$$\Delta = \frac{|\Phi(t_n, \mathbf{x})|^2 - |\Phi(t_{n-1}, \mathbf{x})|^2}{|\Phi(t_{n-1}, \mathbf{x})|^2}. \tag{6.1}$$

If $\Delta < 10^{-4}$ for five consecutive time steps (an arbitrary choice, but one which we have observed to be robust in determining equilibrium states), it is assumed that the condensate number is—on average—constant, and that the system has thermalised. We label the thermalisation time as $t_{\text{therm.}}$

We split the atomic densities into three subsets (according to the best practices outlined in § 2.9): 80% of the samples are used to train the neural network (the training dataset), 10% of the samples are used to validate the neural network at every epoch (the validation dataset), and the remaining 10% of the samples are used to test the predictive abilities of the final machine learning model (the testing dataset). The machine learning model is never exposed to the validation or test datasets.

6.1.2 Architecture

We use a convolutional neural network with an image processing and feature extraction pipeline and then a prediction pipeline. See § 2.7 for a general overview. The specific network we use is shown in Figure 6.1. The model converts rich spatial information in the feature extraction pipeline into a tuple of values which we can associate with the chemical potential and temperature of the sample in the prediction pipeline.

Let each atomic density be $\rho(x, y) = |\Phi(t_{\text{therm.}}, x, y)|^2 \in \mathbb{R}^{N\times N}$, where $N = 256$. The feature extraction pipeline consists of three convolutional layers, which produce a set of 12, 24, and 48 feature maps, Ξ_i^ℓ, where ℓ is the layer index and i is the feature

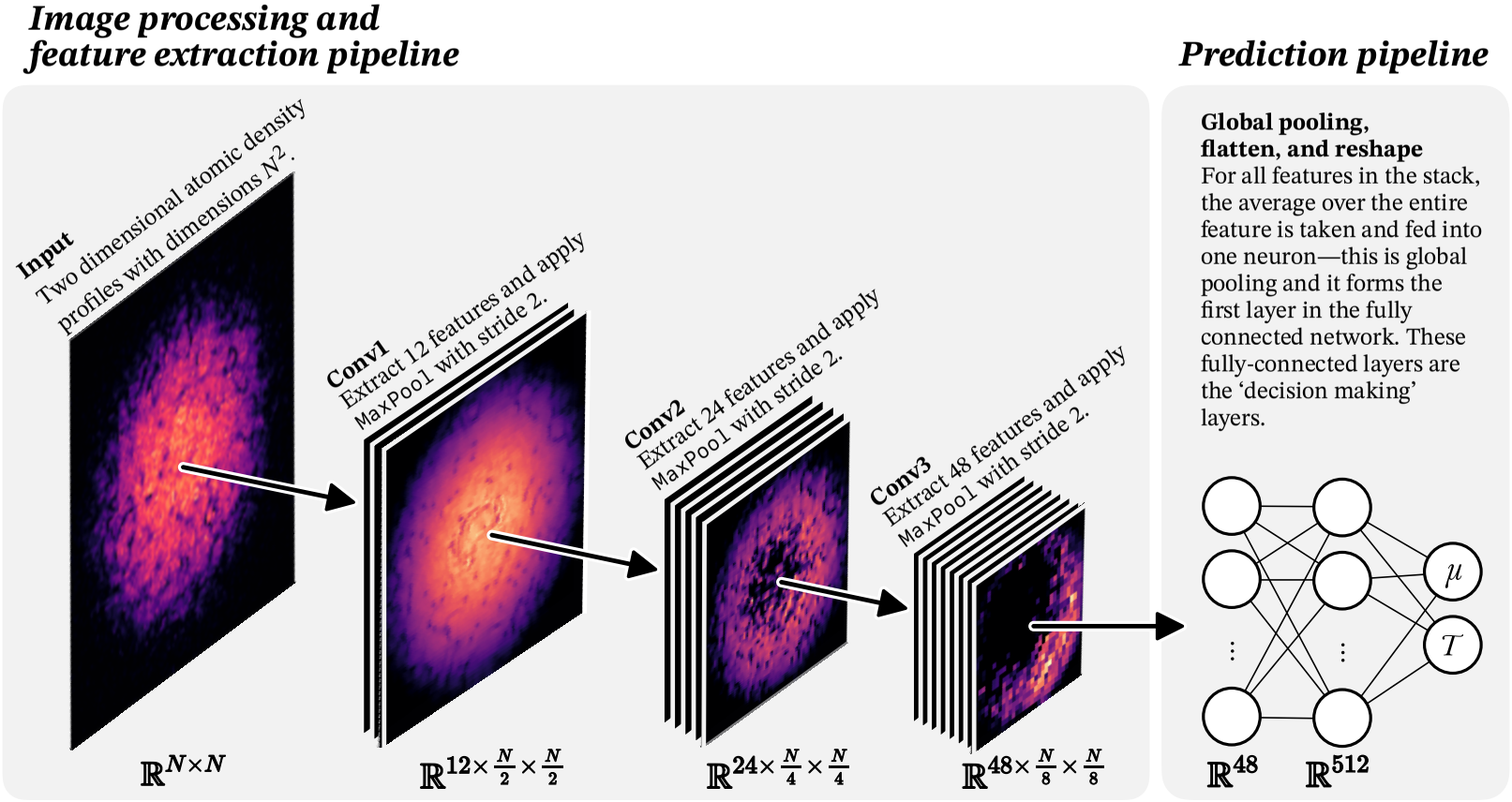


Figure 6.1: Architecture of the convolutional neural network designed to predict the chemical potential, μ, and temperature, T, from atomic density profiles of Bose–Einstein condensates. The network processes 2D density profiles through three convolutional layers (extracting 12, 24, and 48 features, respectively), each followed by a maximum pooling layer with stride 2. Global pooling is applied to each feature before passing through two fully connected layers (FC1 and FC2). ReLU activation functions are used throughout.

index (running from 1 to 12, 24, or 48). Once trained, the feature maps should identify the fingerprints of the chemical potential and temperature in each atomic density sample.

The cost function determines the square error of the predicted and true values of the chemical potential and temperature

$$\mathcal{C} = (\mu_i^{\text{predicted}} - \mu_i^{\text{true}})^2 + (T_i^{\text{predicted}} - T_i^{\text{true}})^2. \tag{6.2}$$

6.1.3 Forward pass

Feature extraction pipeline

The input atomic density is an image $\rho \in \mathbb{R}^{N\times N}$. An example is shown in Figure 6.2 a). The image is processed through three convolutional layers, each followed by ReLU activation and maximum pooling. Figure 6.2 demonstrates intermediate steps to calculate a single feature from the first convolutional layer.

In Layer 1, we extract 12 features. The weights are initialised as $\mathsf{W}_i^1(s,t) \sim U(-1/\sqrt{9}, 1/\sqrt{9})$, using equation (2.77), with size $k_W \times k_W = 3 \times 3$. In the feature extraction pipeline, all weights matrices will be of this size. We perform cross-correlation as follows:

$$\begin{aligned}\zeta_i^1(s_1,t_1) &= (\rho \star \mathsf{W}_i^1)(s_1,t_1) \\ &= \sum_{\sigma=-1}^{1}\sum_{\tau=-1}^{1} \rho(s_1+\sigma,\ t_1+\tau)\,\mathsf{W}_i^1(\sigma,\tau),\end{aligned} \tag{6.3}$$

where $s_1, t_1 \in \{1, 2, \cdots, 256\}$. The cross-correlation of our example atomic density with the final (learnt) weights matrices is shown in Figure 6.2 b). Figure 6.4 shows the output of the cross-correlation of a given atomic density with 12 weights matrices.

We then apply the ReLU activation function with bias:

$$\xi_i^1(s_1,t_1) = \text{ReLU}\big(\zeta_i^1(s_1,t_1) + b_i^1 \mathbb{1}_{N\times N}\big), \tag{6.4}$$

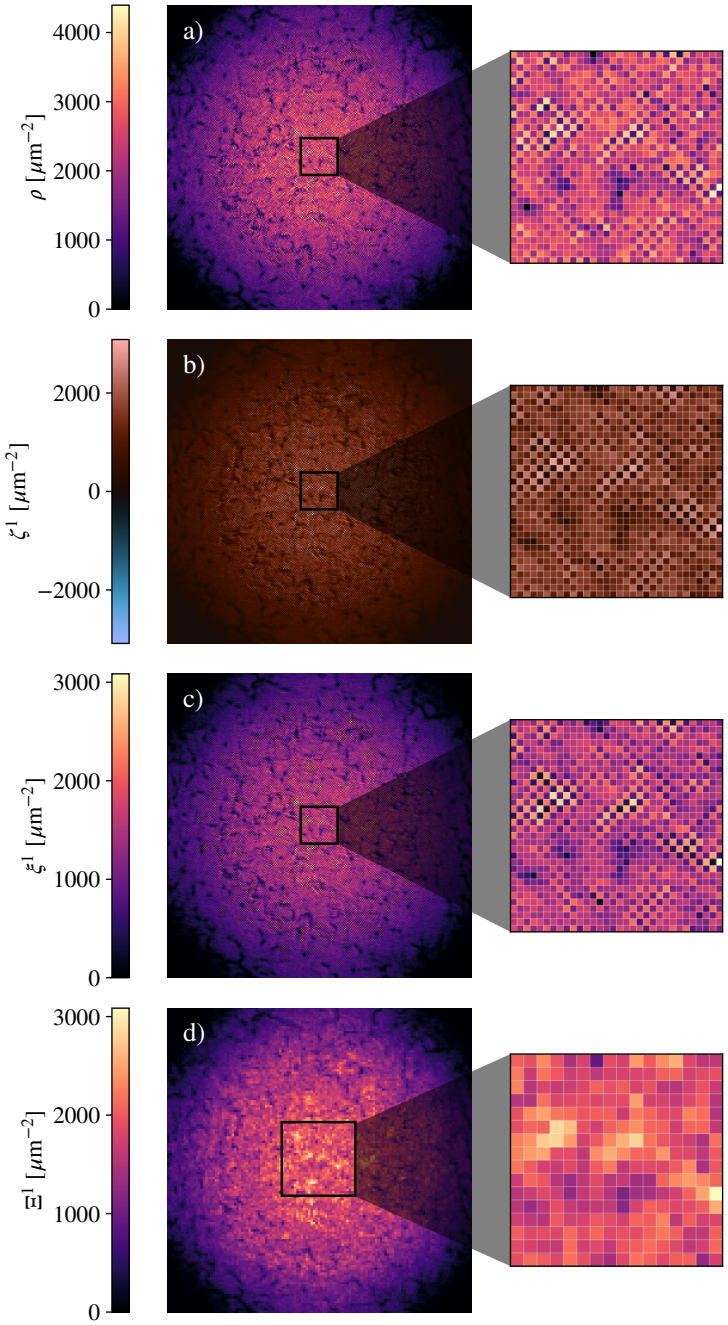


Figure 6.2: a) The atomic density, ρ, is input. b) The cross-correlation of the atomic density with the weights matrices are determined. c) The ReLU activation function and a bias are applied. d) Maximum pooling is performed and the dimensions of the image are halved. This is the output of the first convolutional layer. The right-most figure shows a zoomed-in section as we construct the first layer feature maps (i.e., the output of this layer).

where $\mathbb{1}_{N\times N}$ is a matrix of ones of size $N \times N$. The activation function and bias applied to the output of the cross-correlation is shown in Figure 6.2 c). Figure 6.5 shows the result after applying ReLU activation.

Finally, we apply maximum pooling to obtain the first layer feature maps:

$$\Xi_i^1(s_2, t_2) = \max_{\substack{0\le\sigma<2\\0\le\tau<2}} \xi_i^1(2s_2 - 1 + \sigma,\ 2t_2 - 1 + \tau), \tag{6.5}$$

where $s_2, t_2 \in \{1, 2, \cdots, 128\}$, since the spatial dimensions are halved. These Ξ_i^1 become the feature maps that feed into the next layer. The maximally pooled output is shown in Figure 6.2 d). Figure 6.6 shows the feature maps from the first layer.

In Layer 2, we extract 24 features. The weights are initialised as $\mathsf{W}_i^2(s,t) \sim U(-1/\sqrt{108}, 1/\sqrt{108})$, using equation (2.77). We perform cross-correlation:

$$\begin{aligned}\zeta_i^2(s_2, t_2) &= (\Xi_i^1 \star \mathsf{W}_i^2)(s_2, t_2)\\ &= \sum_{\sigma=-1}^{1}\sum_{\tau=-1}^{1} \Xi_i^1(s_2 + \sigma,\ t_2 + \tau)\, \mathsf{W}_i^2(\sigma, \tau).\end{aligned} \tag{6.6}$$

The output of this cross-correlation is shown in Figure 6.7.

We then apply the ReLU activation function with bias:

$$\xi_i^2(s_2, t_2) = \mathrm{ReLU}\big(\zeta_i^2(s_2, t_2) + b_i^2 \mathbb{1}_{N/2\times N/2}\big). \tag{6.7}$$

Figure 6.8 shows the result after applying ReLU activation.

Finally, we apply maximum pooling to obtain the second layer feature maps:

$$\Xi_i^2(s_3, t_3) = \max_{\substack{0\le\sigma<2\\0\le\tau<2}} \xi_i^2(2s_3 - 1 + \sigma,\ 2t_3 - 1 + \tau), \tag{6.8}$$

where $s_3, t_3 \in \{1, 2, \cdots, 64\}$, since the spatial dimensions are halved again. These Ξ_i^2 become the feature maps that feed into the next layer. Figure 6.9 shows these feature maps.

In Layer 3, we extract the final 48 features which feed into the prediction pipeline. The weights are initialised as $\mathrm{W}_i^3(s,t) \sim U(-1/\sqrt{216}, 1/\sqrt{216})$, using equation (2.77). We perform cross-correlation:

$$\begin{aligned}\zeta_i^3(s_3,t_3) &= (\Xi_i^2 \star \mathrm{W}_i^3)(s_3,t_3) \\ &= \sum_{\sigma=-1}^{1}\sum_{\tau=-1}^{1} \Xi_i^2(s_3+\sigma,\ t_3+\tau)\,\mathrm{W}_i^3(\sigma,\tau).\end{aligned} \tag{6.9}$$

The output of this cross-correlation is shown in Figure 6.10.

We then apply the ReLU activation function with bias:

$$\xi_i^3(s_3,t_3) = \mathrm{ReLU}\big(\zeta_i^3(s_3,t_3) + b_i^3 \mathbb{1}_{N/4\times N/4}\big). \tag{6.10}$$

Figure 6.11 shows the result after applying ReLU activation.

Finally, we apply maximum pooling to obtain the third layer feature maps:

$$\Xi_i^3(s_4,t_4) = \max_{\substack{0\le\sigma<2\\0\le\tau<2}} \xi_i^3(2s_4-1+\sigma,\ 2t_4-1+\tau), \tag{6.11}$$

where $s_4, t_4 \in \{1, 2, \cdots, 32\}$, since the spatial dimensions are halved again. These Ξ_i^3 become the final feature maps that feed into the prediction pipeline. Figure 6.12 shows these feature maps.

Prediction pipeline

The prediction pipeline is a fully connected, feedforward neural network, consisting of three layers, as shown in Figure 6.3. The output of the feature extraction pipeline consists of 48 features Ξ_i^3, which are 32×32 square matrices, i.e., $\Xi_i^3 \in \mathbb{R}^{N/8\times N/8}$, where $i \in \{1, \ldots, 48\}$. Taking the average values over the rows and columns of the 48 features Ξ_i^3 produces 48 reduced feature representations y_i^4. These are the input values for the first layer of the prediction pipeline, and the fourth layer overall.

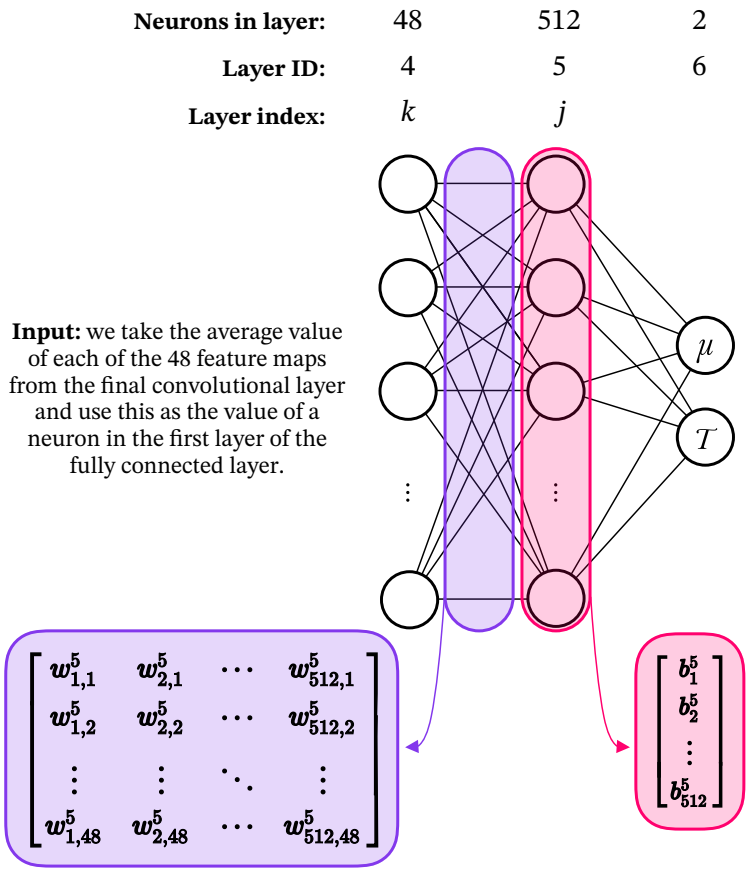


Figure 6.3: The prediction pipeline architecture. Three fully connected, feedforward layers convert the spatial information from the feature extraction pipeline to a tuple of two values associated with the chemical potential and temperature. The weights between two layers and the biases associated with the neurons in a layer are highlighted.

In Layer 5, there are 512 neurons with ReLU activation. The pre-activation values are determined from the y^4_k through

$$z^5_j = \sum_{k=1}^{48} w^5_{jk} y^4_k + b^5_j, \quad j \in \{1, \dots, 512\}. \tag{6.12}$$

We then apply the activation function to produce

$$y^5_j = \mathrm{ReLU}(z^5_j) = \max(0, z^5_j). \tag{6.13}$$

These values feed into Layer 6 (the final layer), through

$$y_j^6 = \sum_{k=1}^{512} w_{jk}^6 y_k^5 + b_j^6, \quad j \in \{1, 2\}, \tag{6.14}$$

which gives predictions for the chemical potential $\mu = y_1^6$ and temperature $T = y_2^6$, in nanokelvin.

We initialise the weights and biases using the Kaiming scheme (equation (2.40)), as $w_{jk}^5,\ b_j^5 \sim U(-1/\sqrt{512}, 1/\sqrt{512})$ and $w_{jk}^6,\ b_j^6 \sim U(-1/\sqrt{2}, 1/\sqrt{2})$.

Training

We use the Adam optimiser (see § 2.5.3) with scalar-valued backpropagation (see § 2.5.4) to update the weights and biases by minimising the cost function in equation (6.2). Backpropagation will give us the new weights and biases to use in the next epoch. Once the cost function in equation (6.2) is sufficiently minimised, we save the weights and biases—this is our model.

When using the model after it has been trained, we re-specify the *architecture* of the neural network (the number of layers, the type of layers, and how many neurons are in each layer) and load the learnt weights and biases into that network. We can then pass new data through our model and obtain predictions of the chemical potential and temperature within fractions of a second.

Figure 6.4: The cross-correlation, ζ^1, of a single-shot atomic density, $\rho(x, y)$, with 12 weights kernels, W^1_i, $i \in \{1, \cdots, 12\}$, as determined by equation (6.3). The cross-correlation may result in positive or negative values, as indicated by the colour bars.

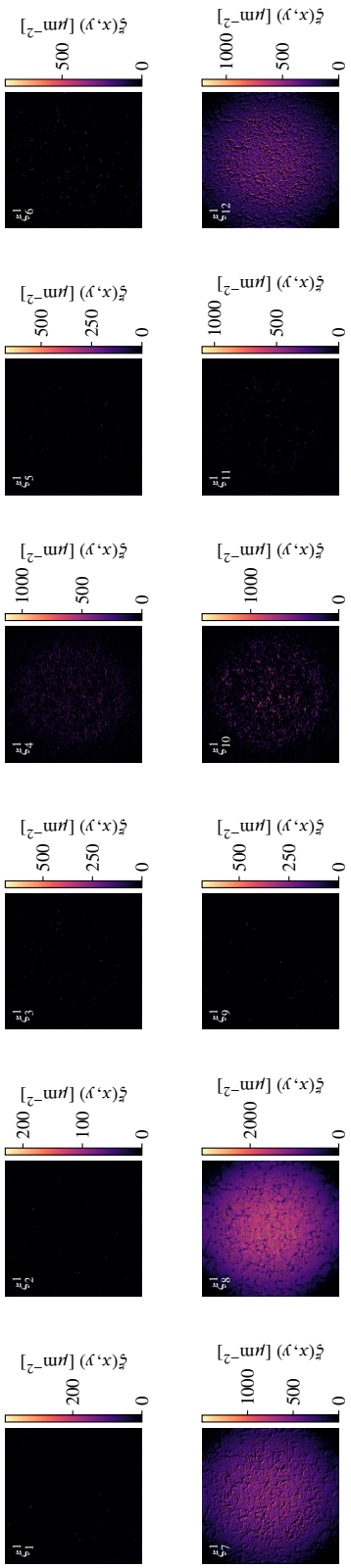


Figure 6.5: ξ^1, the result of applying an activation function as determined by equation (6.4).

Figure 6.6: The maximally pooled features, and therefore the output of the first convolutional layer, Ξ^1, as determined by equation (6.5).

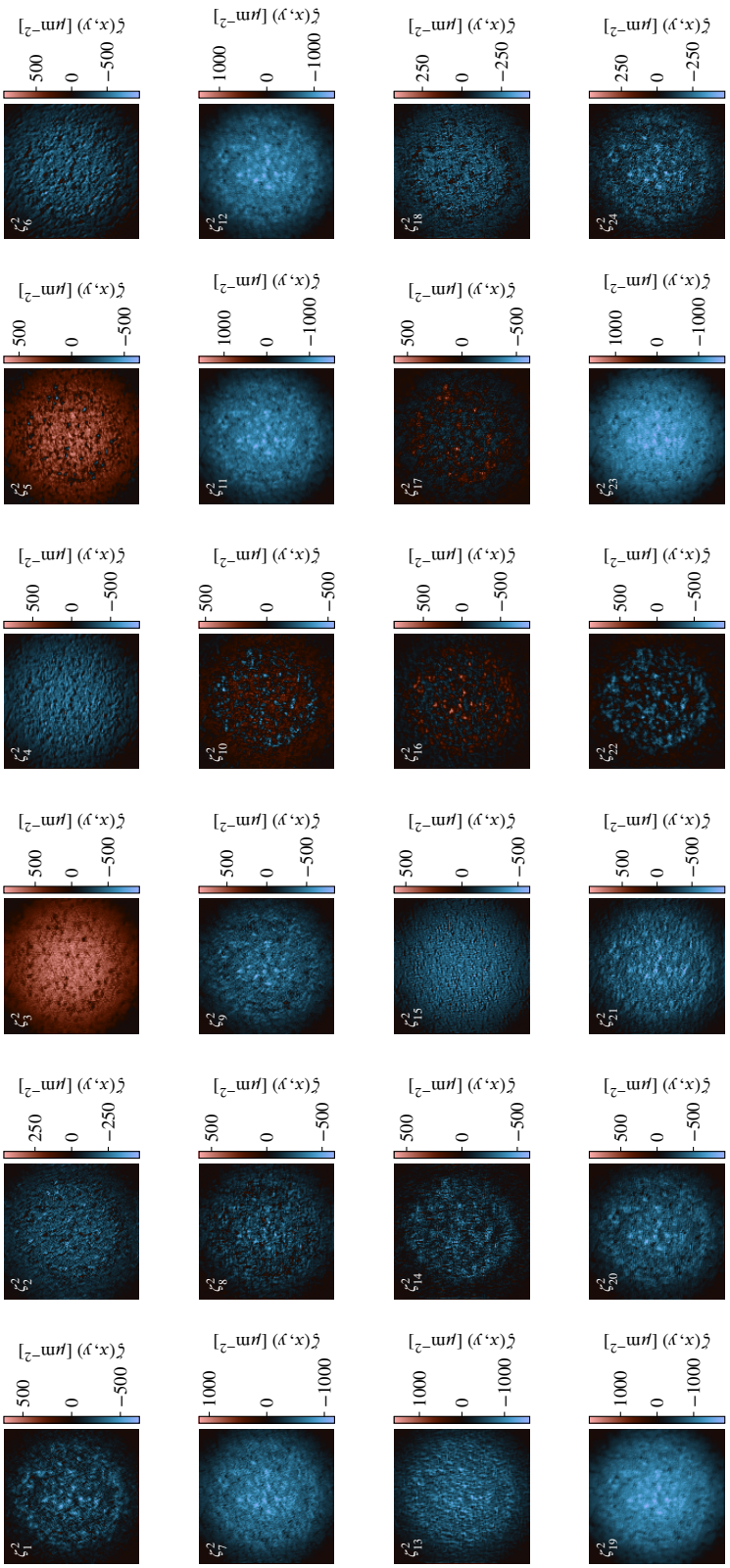


Figure 6.7: The cross-correlation, ζ^2, of the features extracted from the previous layer, Ξ^1, with 24 weights kernels, W_i^2, $i \in \{1, \cdots, 24\}$, as determined by equation (6.6). The cross-correlation may result in positive or negative values, as indicated by the colour bars.

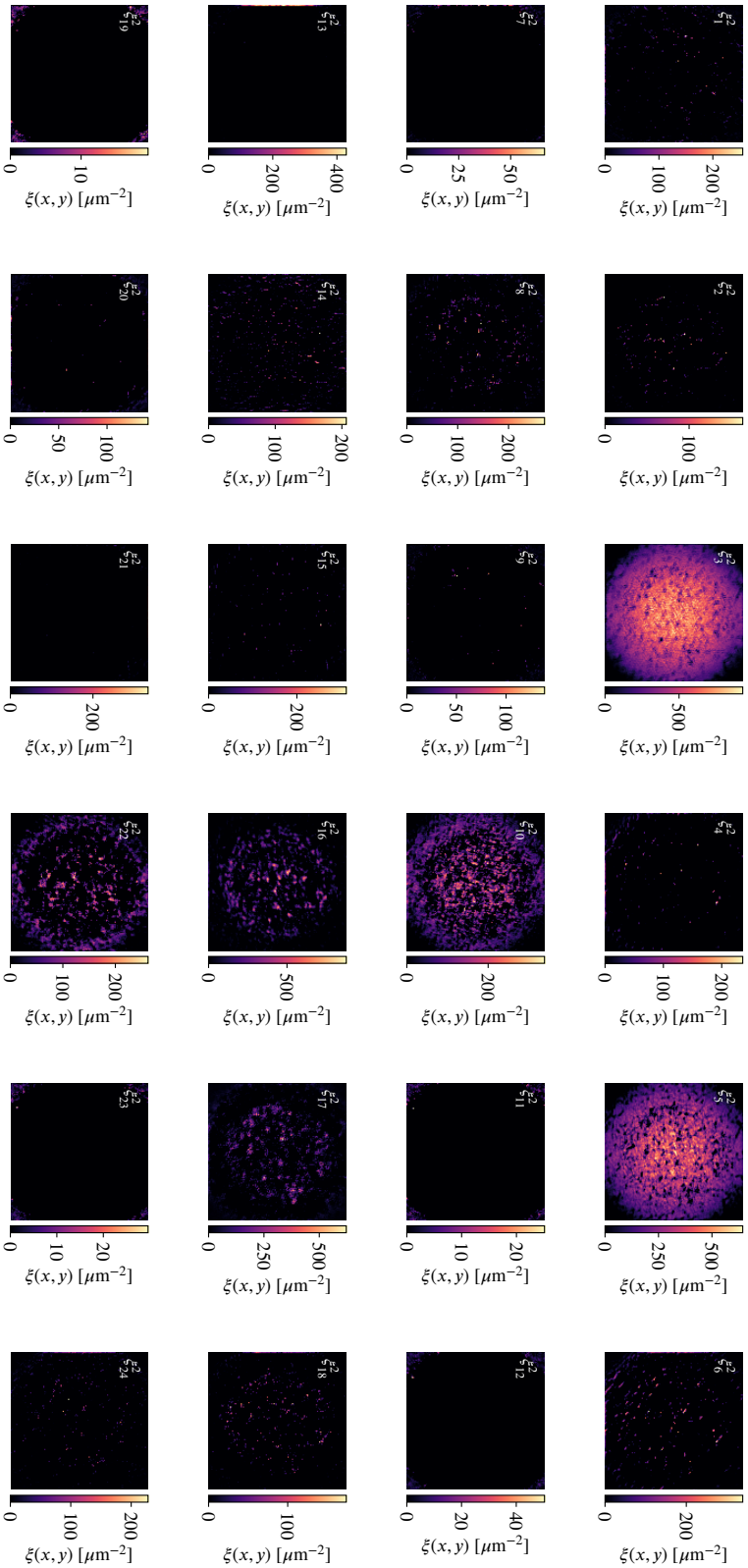


Figure 6.8: The post-activations, ξ^2, as determined by equation (6.7).

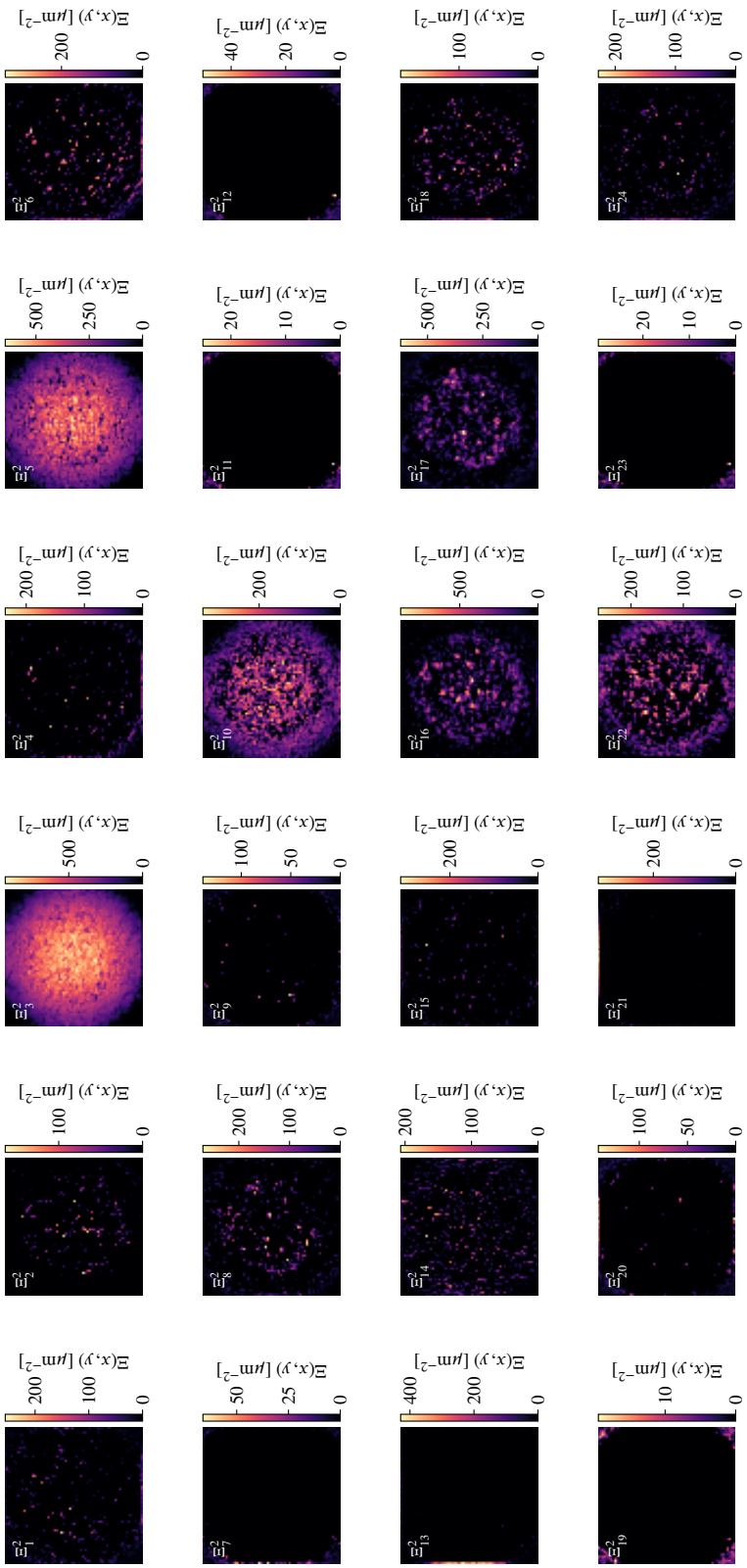


Figure 6.9: The maximally pooled features, and therefore the second layer outputs, Ξ^2, as determined by equation (6.8).

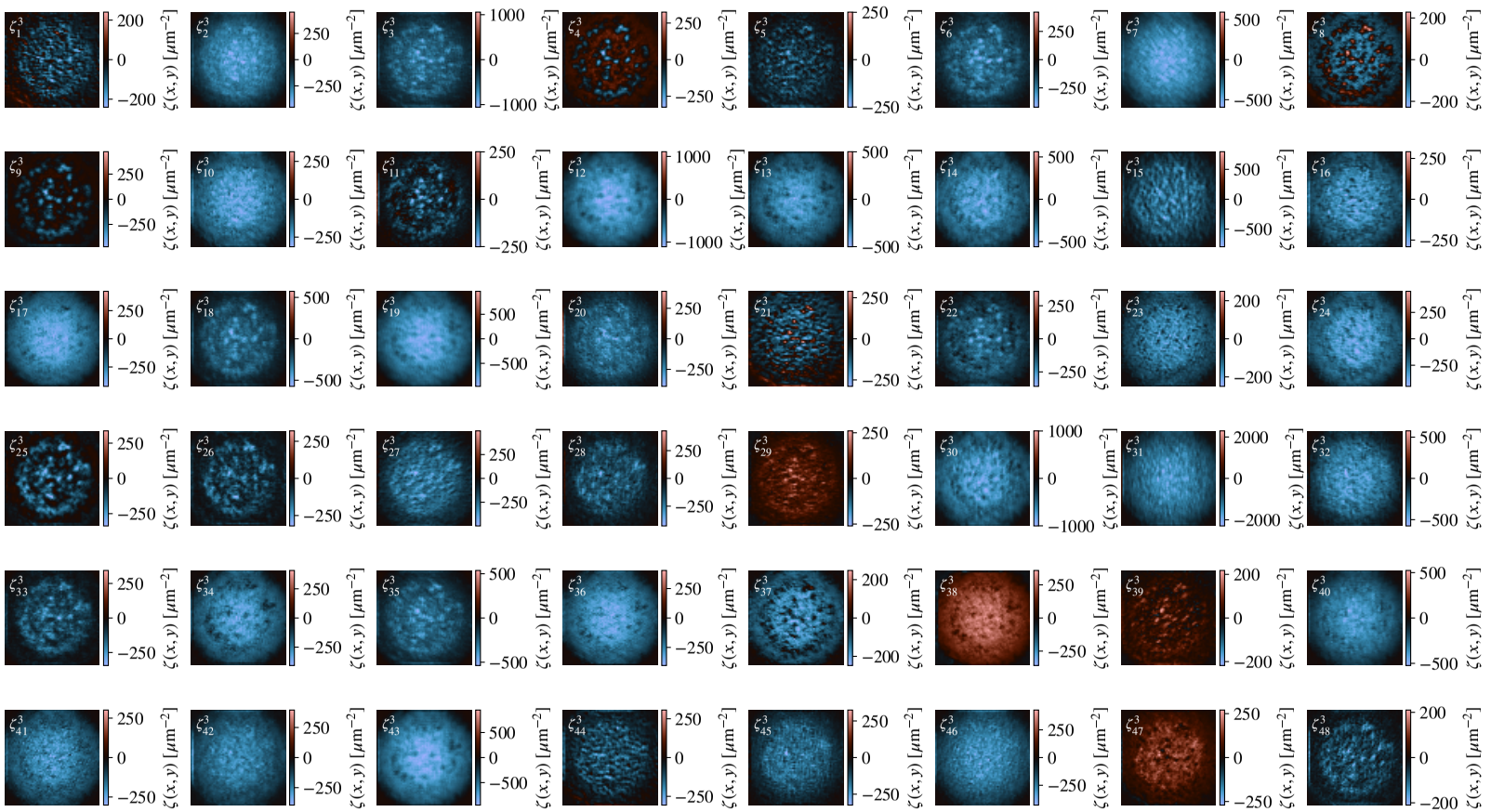


Figure 6.10: The cross-correlation, ζ^3, of the features extracted from the previous layer, Ξ^2, with 48 weights kernels, W_i^3, $i \in \{1, \cdots, 48\}$, as determined by equation (6.9). The cross-correlation may result in positive or negative values, as indicated by the colour bars.

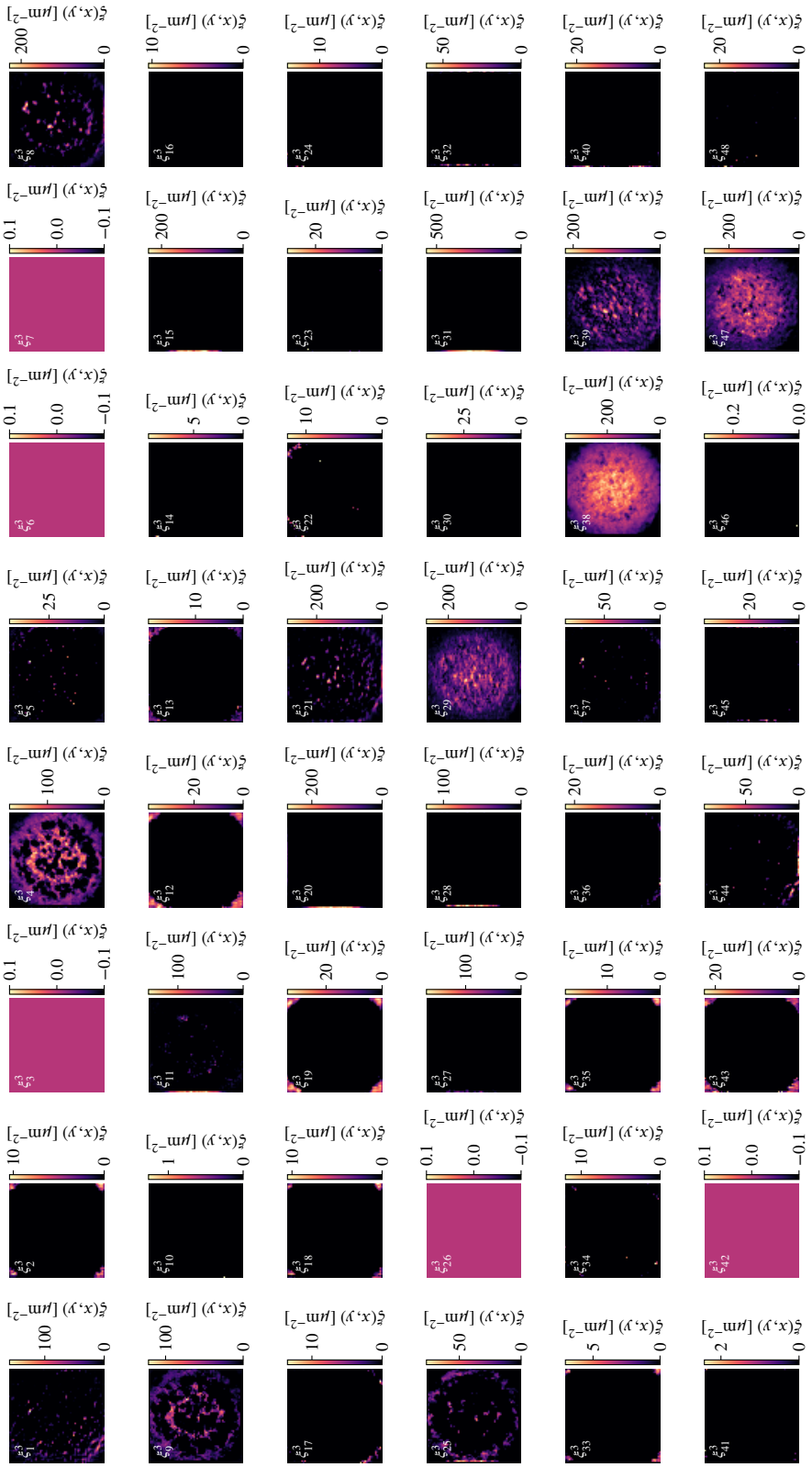


Figure 6.11: The post-activations, ξ^3, as determined by equation (6.10).

Figure 6.12: The maximally pooled features, and therefore the final output of the image processing pipeline, Ξ^3, as determined by equation (6.11).

6.2 Results and discussion

6.2.1 Interpretation of the feature maps

As we progress through the convolutional layers, increasingly abstract feature maps are being learnt. These features may include local variations in the density, patterns or edges corresponding to the trapping potential, recognisable structures such as vortices (see § 3.3.2) or other structures in the atomic cloud. The features may be local or global; the convolutional layers are trying to identify the fingerprints of the chemical potential and temperature in the spatial structure of our input samples.

We observe that the first set of feature maps learn high-density features which may be associated with the chemical potential (since the chemical potential is related to the average atom number). The second set of feature maps learn lower-density features, possibly associated with the edges of the condensates.

The third set of feature maps learn low-density features, which may be associated with thermal fluctuations at this scale (Figure 3.4 demonstrates that the thermal noise is a very low density feature compared to the bulk condensate). The role of temperature in the atomic density is only introduced in the thermal noise, given by the correlation function in equation (3.56); accurate prediction of the temperature using our model is dependent upon being able to effectively capture thermal fluctuations which the convolutional network is attempting to observe in this final convolutional layer.

This hierarchical feature extraction is essential for accurately predicting both the chemical potential and temperature. While these features have the same physical dimensions as the atomic density, they are not directly interpretable as physical quantities due to the applied non-linearities.

After the third convolutional layer, we apply an aggressive global pooling across the 48 feature maps. We then expand the model's representational capacity from 48 to 512 neurons in the first fully connected layer—this layer facilitates complex, non-linear combinations of the pooled features, which is essential for mapping the

subtle details of the density profiles to the desired thermodynamic parameters. We found this layer and the high number of neurons in this layer to be necessary for improving the model's predictive and generalisation capabilities—fewer neurons in this layer was not as conducive to learning (and sufficiently few meant that the network could not predict the values of μ and T within any acceptable error) and more neurons did not appreciably improve the model's predictive capabilities.

Recognising the distinct roles that the chemical potential and temperature have in our feature maps, we design a series of experiments ascertain how our model could be used in a range of experimentally relevant protocols. In particular, we ask: 1) Does the predictive capability of the model depend on the equilibrium distribution, e.g., could we predict the chemical potential and temperature during the thermalisation of a condensate? 2) If the model does not have a strong dependence on the equilibrium distribution, could we use the model on toroidally trapped condensates? 3) Can our model predict the chemical potential and temperature of an ensemble average of equilibrium states?

Let us answer these questions following a brief discussion of the accuracy of the model, which we turn to now.

6.2.2 Model accuracy

In practice, we do not expose the neural network to each atomic density individually because it is computationally inefficient to do so (see our discussions in § 2.5.1 and § 2.7.6). Instead, we send through small batches of atomic densities of size $\beta = 16, 32, 64$ or 128 such that our input is a multidimensional array of size $\rho \in \mathbb{R}^{\beta\times 1\times N\times N}$ and subsequent layers are of size $\rho \in \mathbb{R}^{\beta\times F_\ell\times N\times N}$ (depending on the number of features F_ℓ in that layer). We select β atomic densities randomly and without replacement and feed these through the neural network. We repeat this process until all atomic densities in the sample have been exposed to the neural network. This means that the output of the machine learning model is a vector of β values of μ and a separate vector of β values of T.

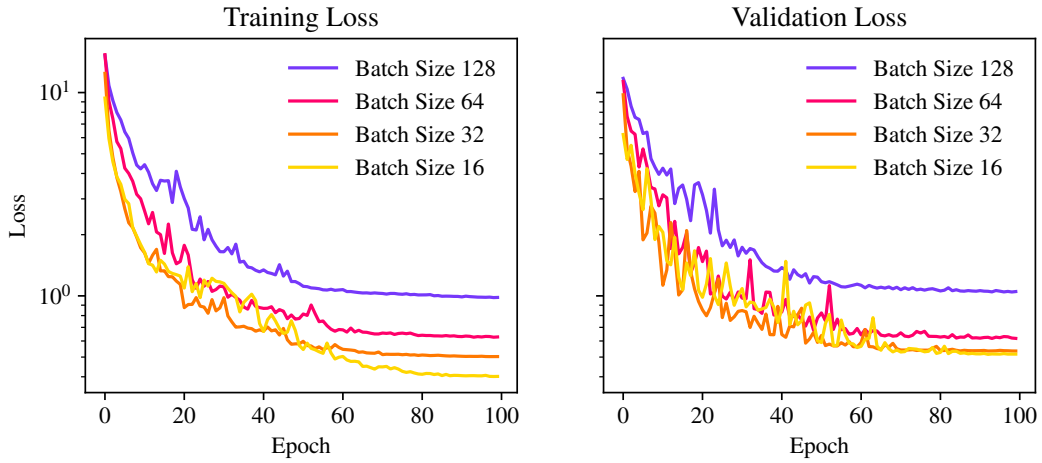


Figure 6.13: The training and validation losses for batch sizes 16, 32, 64, and 128 from a model extracting 12, 24 and 48 features in the first, second, and third convolutional layers. A smaller batch size results in models with a lower overall training and validation loss.

The weight matrices in our neural network are shared across all items in the batch—we apply the same weight matrix to each of the β inputs in the batch. This significantly reduces the number of parameters in our network compared to having separate weights for each input, enhancing the model's capacity to generalise and mitigating overfitting. Furthermore, this approach allows for efficient parallel computation on GPUs, as the same operations are applied simultaneously to all elements in the batch.

We consider 12 models which extract a different number of features per model and use either 16, 32, 64, or 128 batch sizes. One set of models extracts 6, 12, and then 24 features in the first, second, and third convolutional layers. Another set extracts 12, 24, and then 48 features in the first, second, and third convolutional layers. A final set extracts 16, 32, and then 64 features in the first, second, and third convolutional layers. We train each model for 100 epochs, although early-stopping at around 60 epochs was *a posteriori* possible in all models.

Figure 6.13 shows the training and validation losses for the models which extract 12, 24 and 48 features in the convolutional layers over a range of batch sizes. There

is no divergence in the validation loss away from the training loss, so the model does not show overfitting from this metric (see our discussion in § 2.9.3). There was no appreciable difference in the training or validation losses for all models (with fewer or more features)—they all approached final losses of the order of 10^{-1}. It is, however, insufficient to ascertain the accuracy of the model based on these metrics alone.

Figure 6.14 demonstrates the predictive capabilities of the machine learning models which extract 12, 24 and 48 features in the convolutional layers for batch sizes $\beta = \{16, 32, 64, 128\}$. The standard deviation of the absolute errors decreases as the batch size decreases—the model is more accurately predicting the chemical potential and temperature for the smallest batch size. Smaller batch sizes generally lead to better generalisation [Kes+17] by introducing more stochasticity in the optimisation algorithms. This helps the model escape shallow local minima and find more robust solutions that generalise better to unseen data. We define the accuracy within 5% to be the number of samples whose predictions are within 5% of the true values.

Reducing the number of features extracted in the convolutional layers to 6, 12 and 24 has no appreciable impact on predicting the chemical potential (the accuracy within 5% remains between 0.98 and 1.00), but has significant impact on predicting the temperature (the accuracy within 5% lowers to at worst 0.73 and at best 0.92). Increasing the number of features extracted in the convolutional layers to 16, 32, and 64 has no impact on predicting the chemical potential nor the temperature (when the accuracy within 5% is determined to two significant figures). The best model—that which is most computationally efficient and accurate—is the one which extracts 12, 24 and 48 features and uses a batch size of 16. All analysis going forward will use this model.

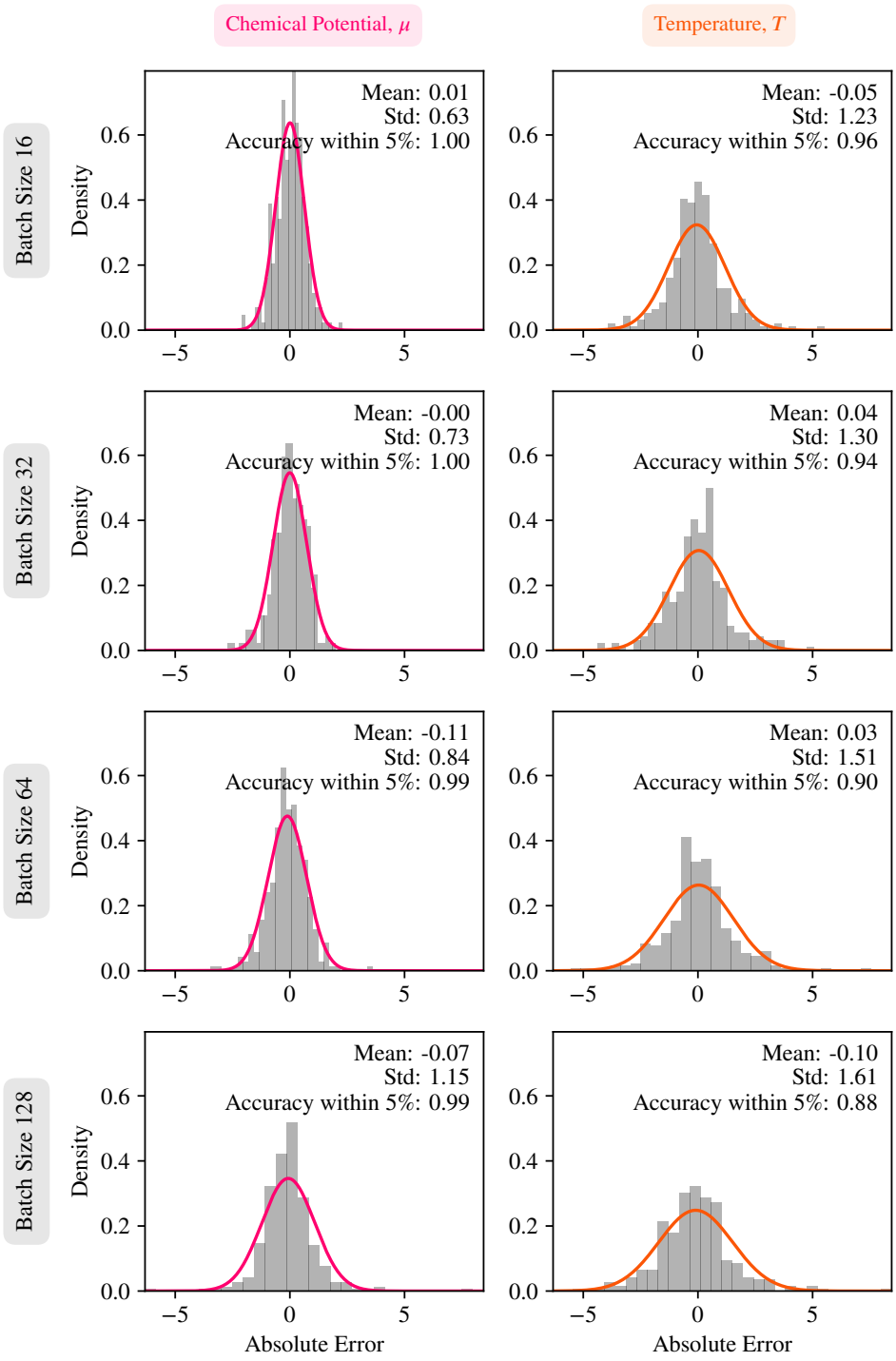


Figure 6.14: The histograms and corresponding normal distributions demonstrate the absolute error in the model's prediction of the chemical potential (column 1) and temperature (column 2) to the true values input to our SGPE simulations. The plots share a common x-axis for easier comparison. These plots correspond to the models which extract 12, 24 and 48 features.

6.2.3 Prediction capability during thermalisation

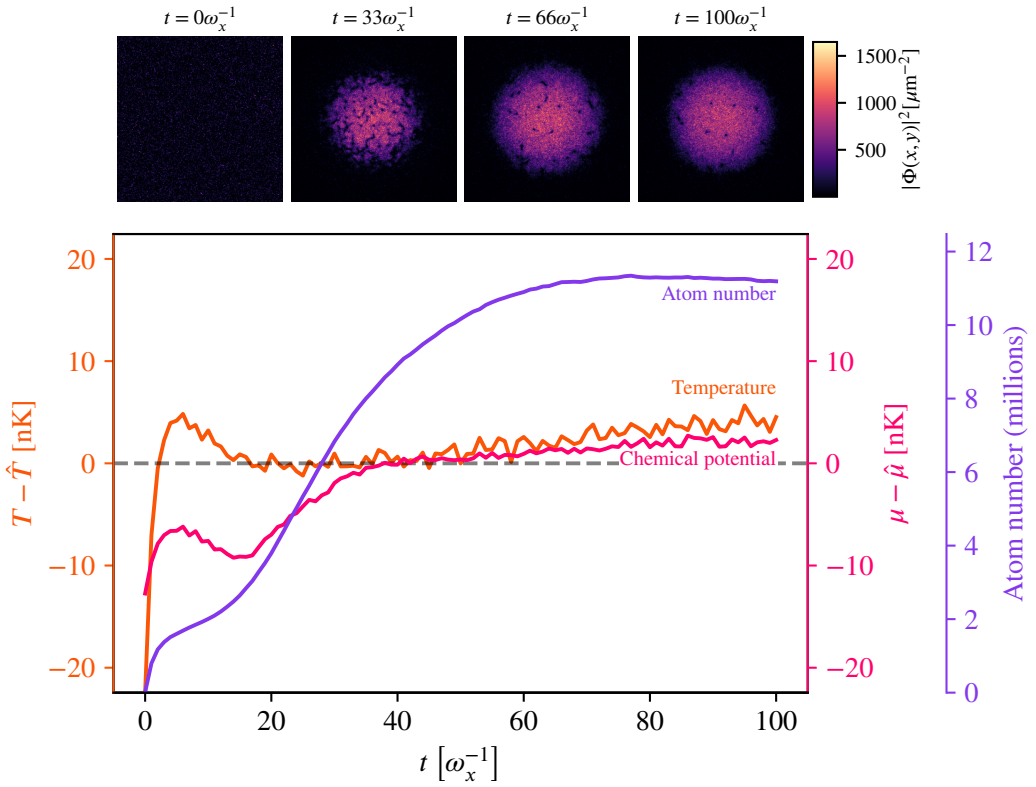


Figure 6.15: The evolution of the prediction of the chemical potential and temperature over the thermalisation procedure on dual axes. The square data points are the absolute errors in the chemical potential. The circular data points are the absolute errors in the temperature. Inset: evolution of a harmonically trapped condensate. The temperature of the same is 22 nK. The chemical potential of the sample is also 22 nK. The absolute errors in the temperature and chemical potential plateau after 80 units of rescaled time.

In our simulations, we do not implement a quench protocol of the chemical potential or the temperature; the thermal bath is consistently parameterised by a constant chemical potential, μ, and temperature, T. Furthermore, the machine learning model is only trained on thermalised Bose gases. Despite this, the model displays excellent accuracy once the system has evolved for a few time steps beyond its initially noisy state, even before the state has fully thermalised. After sufficient

evolution, the model is able to predict the chemical potential and temperature to within a few nanokelvin. Figure 6.15 shows the predictive capabilities of the model during thermalisation.

The model's ability to predict parameters during thermalisation, despite training only on equilibrium states, suggests it has learned features that are indicative of the system's thermodynamic state even out of equilibrium. This unexpected capability requires further investigation into what physical information the network is extracting. This is a useful observation which may allow experimentalists to ascertain whether thermal equilibrium has been reached.

6.2.4 Prediction from toroidally trapped condensates

Toroidally trapped condensates permit metastable persistent currents (quantised flow of Bose–Einstein condensates in a multiply connected geometry (see § 3.3.2)) [Ryu+07] and provide an appropriate geometry for experimental protocols such as weak links [Ram+11; Wri+13] which are used in several atomtronic devices such as rotational sensors.

Without being trained on toroidally trapped condensates, the machine learning model is able to predict the chemical potential and temperature to within the same accuracy as the harmonically trapped condensate. This suggests a degree of generalisability which may arise from the convolutional layer picking up on local and global features. We speculate whether the machine learning model has understood that density depletions due to vortices have no significant impact on the temperature or chemical potential of the condensate. Since the toroidal condensate features a large density depletion at the centre of the torus, the machine learning model may have recognised that this is an unimportant feature to predict the chemical potential and temperature. The results suggest that the model has understood the role of thermal fluctuations in predicting the temperature.

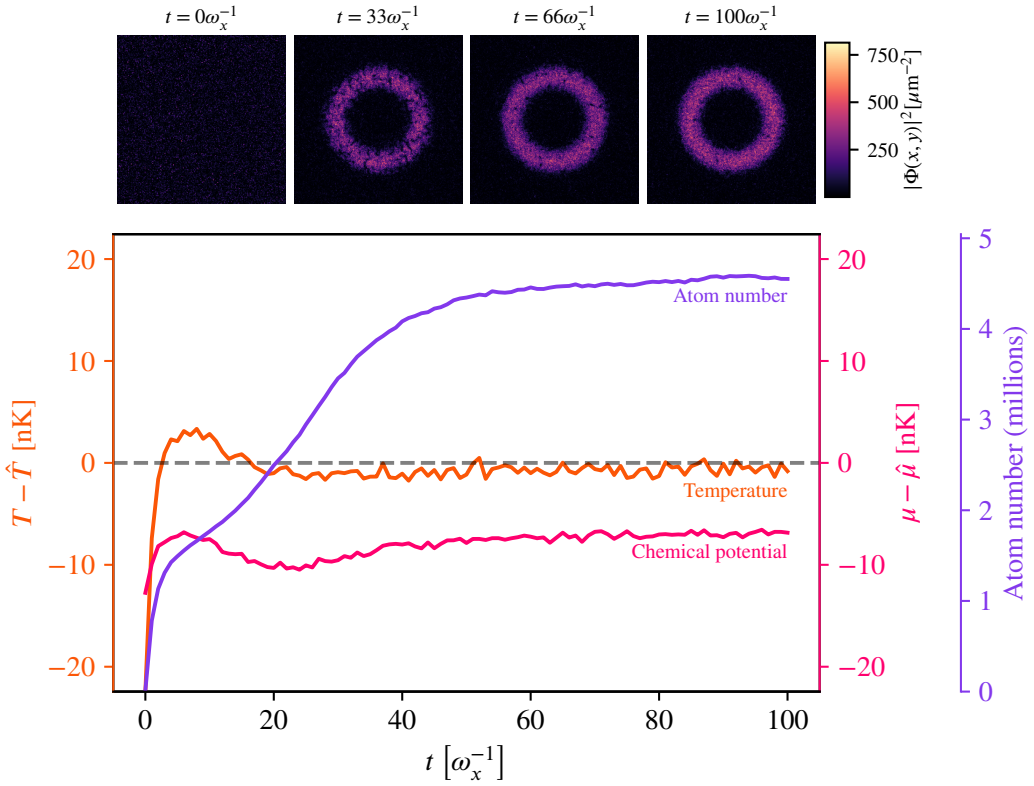


Figure 6.16: Top: density plots of a toroidally trapped condensate during thermalisation. Bottom: evolution of the prediction of the chemical potential and temperature of a toroidally trapped condensate over the thermalisation procedure. The square (pink) data points are the absolute errors in the chemical potential. The circular (orange) data points are the absolute errors in the temperature. The triangular (purple) data points are the atom number. The temperature of the same is 22 nK. The chemical potential of the sample is also 22 nK. The depth of the trap is 60 nK. The minor and major radii are, respectively, 20 μmand 40 μm.

We trap atoms using two-dimensional toroidal potential

$$V(x,y) = V_0\left\{1 - \exp\left[-\frac{1}{\sigma^2}(\rho(x,y) - R)^2\right]\right\}, \tag{6.15}$$

where the trap depth $V_0 = 60\,\mathrm{nK}$, minor radius $\sigma = 20\,\mu\mathrm{m}$ and major radius $R = 40\,\mu\mathrm{m}$. We choose V_0 such that $V_0/\mu > 1$.

6.2.5 Prediction from an ensemble average

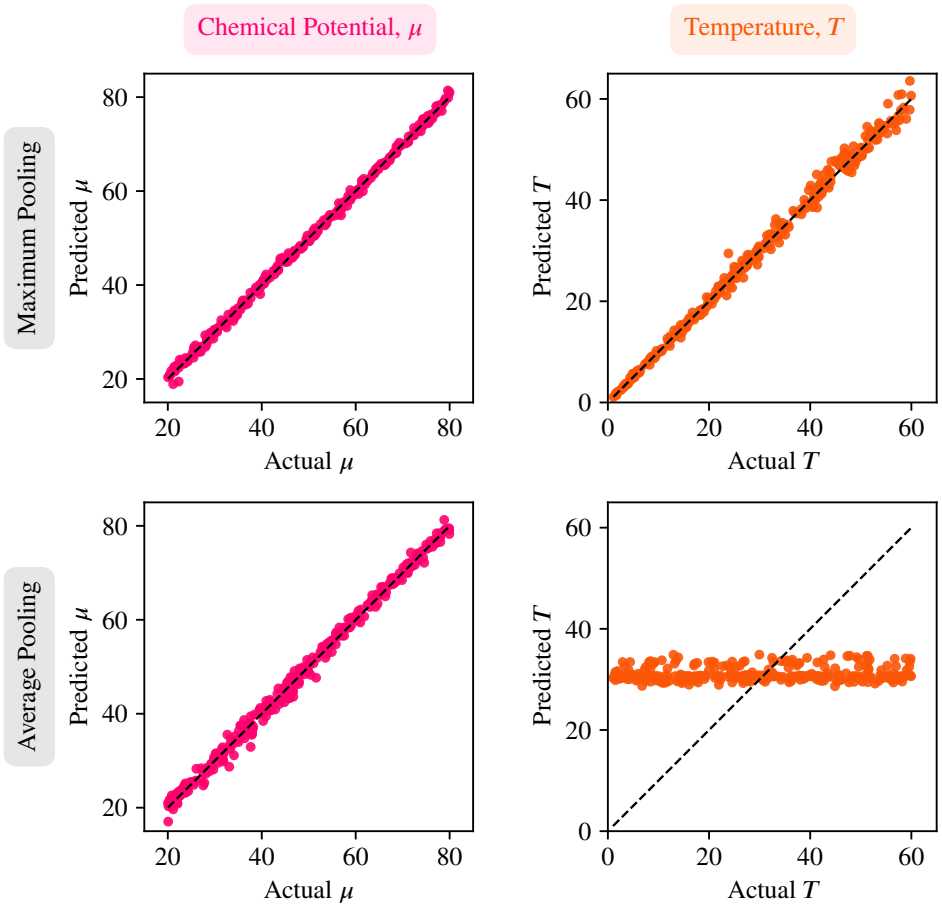


Figure 6.17: A comparison of the predictive capabilities of a network trained with maximum pooling and a network trained with average pooling. Both maximum pooling and average pooling are appropriate for learning and predicting the chemical potential of Bose gases, but only maximum pooling is appropriate for learning and predicting their temperature.

An ensemble average (an average over several noise trajectories) of thermalised states is required to measure any quantum mechanical observable as a function of time, such as the atomic density (by an appropriate correlation function). Akin to experiments, however, numerical runs obtained from a single noise realisation hold important physical information. Indeed, one may interpret a single numerical run of the stochastic Gross–Pitaevskii equation as an independent experimental realisation

[DS01; Wei+08; Coc+10; PDG13]. Averaging over many trajectories washes out the temperature-dependent background fluctuations, and we observe that ensemble averages passed through our machine learning model can only be used to obtain the chemical potential and not the temperature of the sample.

This observation may also be understood in terms of using *average pooling* rather than maximum pooling in the convolutional layers. As discussed in § 2.7, average pooling gives a general representation of smaller regions in the image whereas maximum pooling emphasises the most prominent features in those regions. Figure 6.17 demonstrates that the machine learning model is unable to predict the temperature of an atomic sample if average pooling is used with such a pooling scheme—it is almost certain that the thermal noise has been washed out. This aligns with our empirical observation that the temperature of an ensemble of trajectories cannot be predicted.

6.3 Experimental considerations

To establish an effective pixel size in a typical experiment, we consider the imaging system of Wilson, *et al.* [Wil+15], which—like our model—considers *in situ* imaging of condensates to obtain position distributions. Wilson, *et al.* use a Point Grey Firefly MV CMOS camera with pixels of size $6.0\,\mu\text{m} \times 6.0\,\mu\text{m}$ with a magnification of around 19.7, which leads to an effective pixel size of $6.0\,\mu\text{m}/19.7 = 0.31\,\mu\text{m}$ per pixel. We align our model's resolution to this effective pixel size. For a spherically harmonic condensate of diameter $80\,\mu\text{m}$, we need to use a grid size of 256×256 in order to achieve an effective pixel size of about $0.31\,\mu\text{m}$.

Whilst our model's resolution is well-suited for experimental data analysis, real BEC images often include additional complexities such as imaging artefacts and focus variations. To enhance our model's robustness for experimental use, we might consider augmenting our training data to mimic experimental variations in condensates by, for example, applying an appropriate image filter to mimic imperfections in absorption imaging.

Our model is only trained on *in situ* imaged position distributions. It should in principle be possible to train a model on an expanding time-of-flight velocity distribution, but we do not consider this in this thesis.

6.4 Summary

We have introduced a proof-of-principle machine learning model which can accurately, non-destructively and rapidly predict the chemical potential and temperature of a Bose-condensed cloud of atoms. We use convolutional neural networks—the foundation of most image recognition models—to take an atomic density profile and predict important thermodynamic parameters.

We have demonstrated that our model can accurately predict the chemical potential and temperature for systems which it has not previously been trained on, including during thermalisation and toroidally trapped condensates.

This work joins recent applications of machine learning in the quantum fluids literature, such as the identification of topological defects like vortices [Met+21] and solitons [Guo+21], and the reconstruction of vortex filaments [Kee+23].

Part IV

Conclusions

7

CONCLUSIONS

7.1 Context of this work in the machine learning literature

The first research question we posed was: "can we use techniques in AI to solve physics-based initial value problems?" Exploring several dynamical systems which arise in classical mechanics (including those which exhibit highly non-linear or chaotic dynamics), we introduced the *neural initial value problem*.

We observed that only certain activation functions—belonging to a class which we refer to as probabilistic activation functions—are conducive to learning. Since probabilistic activation functions heuristically implement a type of dropout, we argued that these activation functions implement a form of model averaging, and it was our empirical observation that probabilistic activation functions are an important architectural choice to approximate the solutions of initial value problems.

We also introduced coupled neural network and optimisation schemes. Motivated by the need to solve systems of coupled differential equations with the Hénon–Heiles system, we developed a framework which combines the optimisation schemes of two or more models with a common cost function. Since complex-valued initial value problems can be written as a system of real-valued ordinary differential equations, the coupled optimisation scheme can also be used to solve problems in the complex domain (we have demonstrated this for simple complex-valued ordinary differential equations, but have chosen to omit this from the thesis).

Using our framework for probabilistic activation functions, we provide (to the best of our knowledge) the first derivation of the existing ModReLU activation function

in the complex domain. We also derive the cGELU activation function, which, also to the best of our knowledge, has yet to appear in the complex-valued machine learning literature. Whilst direct complex optimisation appears to be problematic, we note that our approach of solving a system of real-valued differential equations still obtains reliable results (although perhaps not as quickly solving one differential equation might be). We leave the discussion of complex-valued optimisation in this thesis to assist future work.

We extended the neural initial value problem to solve Sturm–Liouville problems which also arise in physics. By using a dual optimisation scheme for the eigenstates and the eigenvalues, we demonstrated a new numerical method for the solution of problems such as the time-independent Schrödinger equation in a harmonic trap and the general Legendre equation (which leads naturally to the spherical harmonics). We envisage the neural Sturm–Liouville problem as being more than a complementary technique—it can optimise the eigenvalue from above or below the physically expected values, it appears to appreciate the discrete spectra of eigenvalues for the problems we have studied, and it finds solutions (eigenstates and eigenvalues) to the problems we have studied in around a minute of wall-clock time.

As a result of our work in this field, we have developed a consistent nomenclature for common algorithms in machine learning. We hope that this will bring some order to complexity.

7.1.1 Future directions of study

Our work on the solution of initial value problems with machine learning could be extended in several directions. The representation of solutions of complex-valued initial value problems as neural networks is unexplored in the literature.

Can we reliably obtain solutions to complex-valued field equations (such as the Gross–Pitaevskii equation) using machine learning methods, and do such methods provide extrema of the corresponding action (as was observed with the classical Hénon–Heiles system)? By decomposing the Gross–Pitaevskii equation into, for

example, a Hermite–Gauss basis,[1] and using the methodology presented in § 4.6 to represent a complex ordinary differential equation as a system of real ordinary differential equations, one could train multiple neural networks for each mode of the Gross–Pitaevskii field and combine to obtain its solution. It is expected that there will be significant computational demands for such a project: efficient parallelisation of the training of each mode may need to be explored and there may be significant memory requirements to hold possibly thousands of neural networks corresponding to the real and imaginary part of each mode in the field.

Future work may also be directed towards understanding the non-monotonicity of the action in the optimisation procedure: for example whether the neural network settles temporarily on a different, solution (which one might consider to be an ODE equivalent of 'hallucination' phenomena observed in AI solutions more generally) obeying Hamilton's principle, possibly one which satisfies the contemporaneous approximation for the initial conditions. Another question of principle is whether a strict action minimisation procedure (choosing two endpoints and then trying to find the minimal action between the points) can be implemented using the framework in this chapter.

Current *direct* implementations of complex-valued neural networks (that is to say, complex-valued optimisation with complex-valued weights and biases) appear not to be conducive to learning. PyTorch and Tensorflow offer complex-valued neural networks as *beta* software, and is prone to bugs, errors, and many functions which are not yet implemented.[2] It may be the case that there are further issues with the implementation of complex optimisation algorithms in PyTorch and Tensorflow, leading to the significant observed divergence between our modelling and the expected results. It could be that there is a theoretical issue in the field of complex

[1] To the best of our knowledge, it is an unsolved problem to find the appropriate eigenstates to represent the Gross–Pitaevskii equation with its non-linearity. However, the decomposition of the Gross–Pitaevskii field into a basis of *linear* solutions to the Schrödinger equation in an external harmonic trap is used extensively in the literature [DC03; TCN09].

[2] Indeed, the Adam optimisation algorithm was only updated with complex number support in 2021.

optimisation which has not yet been addressed in the literature. We have made our complex-valued neural initial value problem code available freely for others to explore [Gri24a].

The neural Sturm–Liouville problem provides excellent approximations to the anharmonic oscillator eigenstates and eigenenergies which are consistent with a first- or second-order perturbative approach. A more thorough analysis of the neural Sturm–Liouville problem with higher-order perturbative approaches may be a useful avenue to explore to ascertain the accuracy of this method for perturbed systems. Whether this approach is only suitable for weakly perturbed systems or could be used for more strongly perturbed systems also needs addressing.

The neural Sturm–Liouville method could be extended to model diatomic molecules using a Morse potential, or radial solutions of the Schrödinger equation using a type of Bessel equation.

7.2 Context of this work in the quantum fluids literature

Our proof-of-principle machine learning approach to predicting thermodynamic parameters of Bose–Einstein condensates represents an advancement in the field of quantum fluids. Traditionally, determining the chemical potential and temperature of a condensate has been a challenging and often destructive process, requiring techniques like time-of-flight expansion. Our model offers a non-destructive, rapid alternative that could streamline experimental procedures and data analysis. Our model could accelerate research in areas such as non-equilibrium dynamics, phase transitions, and quantum turbulence in Bose–Einstein condensates. This technique could enable real-time monitoring of condensate thermodynamics during evaporative cooling or quench dynamics, allowing for precise control and study of non-equilibrium phenomena.

7.2.1 Future directions of study

Implementing Bayesian neural networks could provide confidence intervals for the predictions, providing the user of the model with a measure of the prediction's precision in order to make informed decisions in experimental or numerical protocols.

The model could be trained on simulations with a quench protocol for the chemical potential and temperature, since the stochastic Gross–Pitaevskii equation is a suitable theory for describing the dynamics of condensates at and slightly above the phase transition. It should in principle be possible to build a model which can predict these parameters during an experimental quench, particularly since the model was successful in predicting the chemical potential and temperature out of equilibrium (despite not being trained to do so).

The model was trained only on *in situ* position distributions. We do not anticipate there being issues in training a model using expanding time-of-flight velocity distributions, typically seen in experiments.

Our machine learning model's potential extends beyond predicting chemical potential and temperature. The model could be adapted to output additional parameters such as average atom number and scattering length. This capability becomes particularly relevant when considering real-world experimental scenarios, where imaging techniques introduce imperfections and noise into the data. We propose convolving the atomic density profiles with appropriate filters that mimic the effects of experimental imaging systems. This process would introduce realistic noise and blurring, allowing our model to learn from data more closely resembling experimental outputs. Consequently, the model could be trained to predict the atom number in real time even in the presence of imaging artefacts, a capability of significant practical value in experimental settings.

Apart from extending the model to new experimental scenarios, more fundamental research is needed to understand why the model can accurately predict the chemical potential and temperature out of equilibrium and for toroidally trapped condensates, despite never being trained to do so.

Part V

Appendices

A

COMPLEX ANALYSIS FUNDAMENTALS

Complex analysis is a rich subject—this appendix will not cover the many (often surprising) theorems and applications of the subject. Instead, we refer the reader to the excellent book by Stewart and Tall [ST18]. This appendix is the minimal information needed to follow § 2.8.

A.1 Complex differentiation

Consider a complex function f defined on an open set $S \subset \mathbb{C}$. The function, f, is differentiable at a point $z_0 \in S$ with a derivative $f'(z_0) \in \mathbb{C}$ if

$$f'(z_0) = \lim_{z \to z_0} \frac{f(z) - f(z_0)}{z - z_0}. \tag{A.1}$$

If f is differentiable at every point of S then it is said to be differentiable or holomorphic. If f is differentiable at z_0, then f is continuous at z_0.

Consider a complex function $f(z) = u(x, y) + iv(x, y)$, where $z = x + iy$ and $u, v \in \mathbb{R}^2 \to \mathbb{R}$ are functions of $x \in \mathbb{R}$ and $y \in \mathbb{R}$. The derivative of f at z, according to equation (A.1), must be consistent regardless of the direction of approach. Hence,

$$f'(z) = \lim_{h \to 0} \frac{f(z + h) - f(z)}{h} = \lim_{h \to 0} \frac{f(z + ih) - f(z)}{ih}, \tag{A.2}$$

where $h \in \mathbb{R}$. Approaching along the real axis,

$$\begin{aligned} f'(z) &= \lim_{h\to 0} \frac{f(z+h) - f(z)}{h} \\ &= \lim_{h\to 0} \frac{u(x+h, y) - u(x, y) + i(v(x+h, y) - v(x, y))}{h}. \end{aligned} \tag{A.3}$$

Taking the limit as $h \to 0$,

$$f'(z) = \frac{\partial u}{\partial x} + i\frac{\partial v}{\partial x}. \tag{A.4}$$

Approaching along the imaginary axis,

$$\begin{aligned} f'(z) &= \lim_{h\to 0} \frac{f(z+ih) - f(z)}{ih} \\ &= \lim_{h\to 0} \frac{f(x+i(y+h)) - f(x+iy)}{ih}. \end{aligned} \tag{A.5}$$

Taking the limit as $h \to 0$,

$$f'(z) = \frac{\partial v}{\partial y} - i\frac{\partial u}{\partial y}. \tag{A.6}$$

Equating equation (A.4) and equation (A.6), we obtain

$$\frac{\partial u}{\partial x} + i\frac{\partial v}{\partial x} = -i\frac{\partial u}{\partial y} + \frac{\partial v}{\partial y} \tag{A.7}$$

By equating the real and imaginary parts, we arrive at the Cauchy–Riemann equations

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \tag{A.8}$$

$$\frac{\partial v}{\partial x} = -\frac{\partial u}{\partial y} \tag{A.9}$$

These equations must be satisfied for a function $f(z)$ to be differentiable at z in the complex plane, implying that $f(z)$ is holomorphic at that point. A complex function is *entire* if it is holomorphic on the whole complex plane. Most elementary functions such as the trigonometric functions, the exponential function and all polynomial functions are holomorphic.

A.2 Wirtinger calculus

Consider again the complex function $f(z) : \mathbb{C} \to \mathbb{C}$, where the complex number $z = x + iy \in \mathbb{C}$, and where $x, y \in \mathbb{R}$. It is possible to express x and y in terms of z and its complex conjugate, z^*,

$$x = \frac{1}{2}(z + z^*) \tag{A.10a}$$

$$y = \frac{1}{2i}(z - z^*). \tag{A.10b}$$

By the chain rule,

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial z} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial z} \tag{A.11a}$$

$$= \frac{\partial f}{\partial x} \cdot \frac{1}{2} + \frac{\partial f}{\partial y} \cdot \left(-\frac{i}{2}\right) \tag{A.11b}$$

$$= \frac{1}{2}\left(\frac{\partial f}{\partial x} - i\frac{\partial f}{\partial y}\right). \tag{A.11c}$$

By corollary,

$$\frac{\partial f}{\partial z^*} = \frac{1}{2}\left(\frac{\partial f}{\partial x} + i\frac{\partial f}{\partial y}\right). \tag{A.12}$$

If equation (A.12) is zero and $\frac{\partial f}{\partial z} = \frac{\mathrm{d}f}{\mathrm{d}z} = f'(z)$, then the function f is holomorphic and obeys the Cauchy–Riemann equations.[1]

[1] This is an alternate justification as to why complex-differentiable functions contain no terms in z^*.

B

STOCHASTIC DYNAMICS

Consider a classical particle of mass m immersed in a fluid. By Newton's second law,

$$\mathbf{F} = m\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\lambda\mathbf{v}, \tag{B.1}$$

where λ is a damping coefficient (via Stokes' law) associated with a viscous force. Assuming that the mass of the particle is sufficiently small such that thermal fluctuations in the fluid become important, one can use the equipartition theorem, which relates temperature to the average energies in the system,

$$\frac{1}{2}m\langle\mathbf{v}\rangle = \frac{1}{2}k_B T, \tag{B.2}$$

where the average thermal velocity is

$$v_t = \sqrt{\frac{k_B T}{m}}. \tag{B.3}$$

To capture the thermal velocity in equation (B.1), an additive, random force $m\boldsymbol{\eta}(t)$ is introduced, which describes the collisions between the particle and the molecules of the fluid

$$m\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\lambda\mathbf{v} + m\boldsymbol{\eta}(t). \tag{B.4}$$

Equation (B.4) is referred to as a Langevin equation [Ris12].

Equation (B.4) can be solved as a first-order homogenous differential equation using the integrating factor $\mathrm{e}^{\int \mathrm{d}t\,\gamma}$

$$\mathbf{v}(t) = \mathbf{v}(0)\mathrm{e}^{-\gamma t} + \int_0^t \mathrm{d}\tau\ \mathrm{e}^{-\gamma(t-\tau)}\boldsymbol{\eta}(\tau). \tag{B.5}$$

In order to gain anything useful from this solution, we must average over many realisations of the noise $\eta(t)$. Since the random force arises due to the chaotic motion of molecules in the liquid and their collision with the particle, it is assumed that these forces are uncorrelated at different times. The random forces are also assumed to be drawn from a Gaussian distribution. These two conditions form the *white noise assumption*, and it states that

$$\langle \boldsymbol{\eta}(t) \rangle = \mathbf{0} \quad \text{and} \tag{B.6}$$

$$\left\langle \eta_\mu(t)\eta_\nu(t') \right\rangle = \sigma\delta_{\mu\nu}\delta(t-t'), \tag{B.7}$$

where σ is the strength of the noise, $\delta_{\mu\nu}$ is a Kronecker delta over the spatial (Cartesian) coordinates μ and ν and $\delta(t-t')$ is a Dirac delta function. The values of the strength σ are physically constrained by the equipartition theorem; it can be shown that $\sigma = 2k_B T\lambda$. This is an example of a *fluctuation-dissipation* theorem (the strength of the noise term is proportional to the strength of the dissipations).

C

NUMERICAL INTEGRATION AND DIFFERENTIATION

Runge–Kutta methods are the first tool of choice in a computational physicist's toolkit to numerically integrate a differential equation—they are generally fast (although not always the fastest method), especially when the right-hand side of the differential equation is quickly and efficiently computed.

One must rearrange the differential equation such that the temporal derivative is on the left-hand side and all other terms are on the right-hand side, encapsulated in some arbitrary function $f(t, \Phi(t, \mathbf{x}))$; in principle, this function could be dependent upon derivatives of $\Phi(t, \mathbf{x})$. That is to say, we will only consider differential equations of the form

$$\frac{\partial \Phi(t, \mathbf{x})}{\partial t} = f(t, \Phi(t, \mathbf{x})). \tag{C.1}$$

C.1 Euler method

The Euler method is

$$\Phi_{n+1} = \Phi_n + \Delta t f(t_n, \Phi_n), \tag{C.2}$$

which advances our solution from t_n to $t_{n+1} \equiv t_n + \Delta t$. The error in the Euler method is $\mathcal{O}(\Delta t^2)$.

C.2 Runge–Kutta second-order method

One can use an intermediate (trial) step to the midpoint of the interval $[t_n, t_{n+1}]$ through the routine

$$\begin{aligned} k_1 &= f(t_n, \Phi_n) \\ k_2 &= f\left(t_n + \frac{1}{2}\Delta t, \Phi_n + \frac{1}{2}k_1\right) \\ \Phi_{n+1} &= \Phi_n + \Delta t k_2 + \mathcal{O}(\Delta t^3), \end{aligned} \tag{C.3}$$

which is a second-order method since the symmetrisation about the interval's midpoint cancels out the error proportional to Δt. Equations (C.3) are called the second-order Runge–Kutta method.

C.3 Runge–Kutta fourth-order method

The most common Runge–Kutta method involves evaluating four intermediate steps (compared to the two intermediate steps of the second-order method). Hereafter referred to as the Runge–Kutta fourth-order method, one computes

$$\begin{aligned} k_1 &= f(t_n, \Phi_n) \\ k_2 &= f\left(t_n + \frac{1}{2}\Delta t, \Phi_n + \frac{1}{2}k_1\right) \\ k_3 &= f\left(t_n + \frac{1}{2}\Delta t, \Phi_n + \frac{1}{2}k_2\right) \\ k_4 &= f(t_n + \Delta t, \Phi_n + k_3) \\ \Phi_{n+1} &= \Phi_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(\Delta t^5). \end{aligned} \tag{C.4}$$

C.4 Adaptive Runge–Kutta–Fehlberg fourth-fifth method

Adaptive step-size methods do not require a user-specified time step Δt. The adaptive Runge–Kutta–Fehlberg fourth-fifth method (RKF45) computes a local truncation error between every integration step, checking whether this error is within some user-defined tolerance. The algorithm continuously checks convergence during

integration, decreases the step size for regions with lots of spatial variation, and increases the step size for regions without lots of spatial variation. The algorithm adapts to the system's evolution, unlike fixed-step algorithms which do not. For the same reason, simulations with time-dependent parameters and equations also benefit from adaptive step-size algorithms.

The ARKF45 method is briefly presented here, following the discussion by Press [Pre+92]. For notational clarity later in this description, the time step is denoted as h.

The ARKF45 method describes a fifth-order method $\{\Phi\}$ with six function evaluations $\{f_i\}$ and another combination of the same functions describing a fourth-order method $\{\Phi^*\}$. When combined, the fourth-fifth algorithm describes a numerical integration of $\{\Phi\}$ with some local truncation error Δ. The fifth-order method is

$$\Phi_{n+1} = \Phi_n + \sum_{i=1}^{6} c_i k_i + \mathcal{O}(h^6) \tag{C.5}$$

where $\{c_i\}$ are coefficients determined by Cash and Karp [CK90] and $\{k_i\}$ are the local increment functions in the standard Runge–Kutta algorithm. The local increment functions are defined as

$$\begin{aligned} k_1 &= hf(t_n, \Phi_n) \\ k_2 &= hf(t_n + a_2 h, \Phi_n + b_{21} k_1) \\ &\cdots \\ k_6 &= hf(t_n + a_6 h, \Phi_n + b_{61} k_1 + \cdots + b_{65} k_5) \end{aligned} \tag{C.6}$$

where $\{a_i\}$ and $\{b_{ij}\}$ are further sets of Cash-Karp coefficients and h is the integration step size. The fourth-order method is

$$\Phi^*_{n+1} = \Phi^*_n + \sum_{i=1}^{6} c^*_i k_i + \mathcal{O}(h^5). \tag{C.7}$$

The local truncation error is thus

$$\Delta \equiv \Phi_{n+1} - \phi^*_{n+1} = \sum_{i=1}^{6} (c_i - c_i^*) k_i. \tag{C.8}$$

Consider the step from h_0 to h_1 with associated local truncation errors Δ_0 and Δ_1. Since $\Delta \propto h^5$, the step h_0 must obey

$$h_0 = h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{1/5}. \tag{C.9}$$

Hidden in this mathematical ambiguity lies the summit of the mountainous climb through numerical integrators. If the truncation error $\Delta_1 > \Delta_0$, where Δ_0 is the user-defined tolerance, the ARKF45 algorithm will compute how much to *decrease* the step-size based on the pre-defined tolerance (usually $\sim 10^{-6}$) and retry the *current* integration step. Conversely, if $\Delta_1 < \Delta_0$, the ARKF45 algorithm will compute how much it can safely *increase* the step size based on the tolerance for the *next* integration step.

Despite the additional computation, the performance gains in using adaptive step-sizes for some problems are hundred- or thousand-fold [Pre+92].

C.5 Numerical evaluation of derivatives

The definition of the derivative of a one-dimensional, time-independent wave function is

$$\frac{\mathrm{d}\phi(x)}{\mathrm{d}x} = \lim_{h \to 0} \frac{\phi(x+h) - \phi(x)}{h}, \tag{C.10}$$

where h is an infinitesimal, positive or negative quantity. The right-hand side is, therefore, an approximation to the derivative of ϕ, which increases in accuracy as $h \to 0$. The error of the approximation can be obtained by the Taylor expansion of ϕ

$$\phi(x+h) \approx \phi(x) + h\frac{\mathrm{d}\phi(x)}{\mathrm{d}x} + h^2 \frac{\mathrm{d}^2\phi(x)}{\mathrm{d}x^2}. \tag{C.11}$$

Rearranging equation (C.11) gives

$$\frac{\phi(x+h)-\phi(x)}{h} - \frac{\mathrm{d}\phi(x)}{\mathrm{d}x} \approx h\frac{\mathrm{d}^2\phi(x)}{\mathrm{d}x^2}, \tag{C.12}$$

i.e., the error in the quotient of the definition of the derivative is proportional to h.

If $h = \Delta x > 0$, where Δx is some finite (and not infinitesimal) quantity, then

$$\frac{\mathrm{d}\phi(x)}{\mathrm{d}x} \approx \frac{\phi(x+\Delta x)-\phi(x)}{\Delta x}, \tag{C.13}$$

which is the *forward difference* of the derivative. If $h = \Delta x < 0$, then

$$\frac{\mathrm{d}\phi(x)}{\mathrm{d}x} \approx \frac{\phi(x)-\phi(x-\Delta x)}{\Delta x}, \tag{C.14}$$

which is the *backward difference* of the derivative. Subtracting the forward difference from the backward difference gives the centred difference approximation of the derivative

$$\frac{\mathrm{d}\phi(x)}{\mathrm{d}x} \approx \frac{\phi(x+\Delta x)-\phi(x-\Delta x)}{2\Delta x}, \tag{C.15}$$

which has an error $\mathcal{O}(\Delta x^2)$. Using higher-order terms in the Taylor expansion in equation (C.11) gives higher-order difference formulae. The choice of which order method to use is not *a priori* obvious; we find second-order methods to be sufficient throughout all of our computational work.

Second-order derivatives can be approximated similarly. We will use the approximation

$$\frac{\mathrm{d}^2\phi(x)}{\mathrm{d}x^2} \approx \frac{\phi(x+\Delta x)-2\phi(x)+\phi(x-\Delta x)}{\Delta x^2}, \tag{C.16}$$

which has an error $\mathcal{O}(\Delta x^2)$. For multivariate wave functions, the definition of the Laplacian implies that

$$\begin{aligned}\nabla^2\phi(x,y) &= \frac{\partial^2\phi(x,y)}{\partial x^2} + \frac{\partial^2\phi(x,y)}{\partial y^2} \\ &\approx \frac{\phi(x+\Delta x,y) - 2\phi(x,y) + \phi(x-\Delta x,y)}{\Delta x^2} \\ &\quad + \frac{\phi(x,y+\Delta y) - 2\phi(x,y) + \phi(x,y-\Delta y)}{\Delta y^2}.\end{aligned} \tag{C.17}$$

The implementation of equations (C.16) and (C.17) in Rust are available in our open-source code [Gri24c].

C.6 Solving Sturm–Liouville equations numerically: the shooting method

We first re-write the Sturm–Liouville equation as a system of first-order ordinary differential equations for compatibility with the numerical integration routines in appendix C

$$\frac{d\phi}{dx} = y, \tag{C.18}$$

$$\frac{dy}{dx} = \frac{q(x) - \lambda w(x)}{p(x)}\phi - \frac{p'(x)}{p(x)}y. \tag{C.19}$$

One of the boundary conditions is used as an initial condition for the system of differential equations. The other boundary condition becomes the target condition. The eigenvalue λ is iteratively updated based on the *shooting method*. Akin to firing a cannon at a target, the shooting method iteratively corrects a shooting parameter (e.g., an initial velocity, or more generally the eigenvalue λ) based on how far away the solution is from the target (the other boundary condition).

D

CROSS CORRELATION AS A FROBENIUS INNER PRODUCT

We show that the cross-correlation operation, as defined in equation (2.76), can be elegantly interpreted as a trace operation and therefore a Frobenius inner product.

We can reframe the cross-correlation as a series of matrix multiplications and summations where, for each position (s, t), we effectively compute the sum of element-wise products between a sub-matrix of X and the weights matrix/kernel W.

Recall that for two matrices A and B of compatible dimensions, the trace of the matrix product AB

$$\operatorname{tr}(\mathsf{AB}) = \sum_{i,j} \mathsf{A}_{ij}\mathsf{B}_{ji}. \tag{D.1}$$

Let us define $\mathsf{X}_{s,t}$ as the sub-matrix of X centred at position (s, t) with the same dimensions as W. We can then express the cross-correlation at position (s, t) as

$$(\mathsf{X} \star \mathsf{W})(s, t) = \operatorname{tr}(\mathsf{X}_{s,t}\mathsf{W}^T) \tag{D.2}$$

The Frobenius inner product is

$$\langle \mathsf{A}, \mathsf{B} \rangle_F = \operatorname{tr}(\mathsf{AB}^T) = \sum_{i,j} \mathsf{A}_{ij}\mathsf{B}_{ij}, \tag{D.3}$$

which leads to the definition of the cross-correlation operation in terms of perhaps simpler linear algebraic operations

$$(\mathsf{X} \star \mathsf{W})(s,t) = \langle \mathsf{X}_{s,t}, \mathsf{W} \rangle_F. \tag{D.4}$$

Index

Bibliography

[HH64] M. Hénon, C. Heiles. *The applicability of the third integral of motion: Some numerical experiments*. Astron. J. **69** (1964).

[Hub64] P. J. Huber. *Robust Estimation of a Location Parameter*. Ann. Math. Stat **35**.1 (1964).

[Kel64] L. V. Keldysh. *Diagram Technique for Non-Equilibrium Processes*. J. Exptl. Theoret. Phys. (U.S.S.R.) **47** (1964).

[Fuk69] K. Fukushima. *Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements*. IEEE Trans. Pattern Anal. Mach. **5**.4 (1969).

[Fet72] A. L. Fetter. *Nonuniform states of an imperfect bose gas*. Ann. Phys. (N. Y.) **70**.1 (1972).

[Fuk80] K. Fukushima. *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*. Biol Cybern. **36**.4 (1980).

[Vap82] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, Berlin, 1982.

[HSW89] K. Hornik, M. Stinchcombe, H. White. *Multilayer feedforward networks are universal approximators*. Neural Netw. **2**.5 (1989).

[LeC+89] Y. LeCun et al. *Backpropagation Applied to Handwritten Zip Code Recognition*. Neural Comput. **1**.4 (1989).

[CK90] J. R. Cash, A. H. Karp. *A Variable Order Runge-Kutta Method for Initial Value Problems with Rapidly Varying Right-Hand Sides*. ACM Trans. Math. Softw. **16**.3 (1990).

[HS90] L. Hansen, P. Salamon. *Neural network ensembles*. IEEE Trans. Pattern Anal. Mach. **12**.10 (1990).

[Pre+92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery. *Numerical Recipes in C*. Second. Cambridge, USA: Cambridge University Press, 1992.

[Les+93] M. Leshno, V. Y. Lin, A. Pinkus, S. Schocken. *Multilayer feedforward networks with a nonpolynomial activation function can approximate any function*. Neural Netw. **6**.6 (1993).

[KV94] A. Krogh, J. Vedelsby. *Neural Network Ensembles, Cross Validation, and Active Learning*. *Adv. Neural Inf. Process. Syst. 32*. Ed. by G. Tesauro, D. Touretzky, T. Leen. Vol. 7. MIT Press, 1994.

[And+95] M. H. Anderson et al. *Observation of Bose-Einstein Condensation in a Dilute Atomic Vapor*. Science **269**.5221 (1995).

[Bra+95] C. C. Bradley, C. A. Sackett, J. J. Tollett, R. G. Hulet. *Evidence of Bose-Einstein Condensation in an Atomic Gas with Attractive Interactions*. Phys. Rev. Lett. **75** (1995).

[Dav+95] K. B. Davis et al. *Bose-Einstein Condensation in a Gas of Sodium Atoms*. Phys. Rev. Lett. **75** (1995).

[LeC+95a] Y. LeCun et al. *Comparison of learning algorithms for handwritten digit recognition*. *International Conference on Artificial Neural Networks, Paris*. Ed. by F. Fogelman, P. Gallinari. EC2 Cie, 1995.

[LeC+95b] Y. LeCun et al. *Learning Algorithms for Classification: A Comparison on Handwritten Digit Recognition*. *Neural Networks: The Statistical Mechanics Perspective*. World Scientific, 1995.

[Vap95] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer New York, 1995.

[Bre96] L. Breiman. *Bagging predictors*. Mach. Learn. **24**.2 (1996).

[Bur96] K. Burnett. *Bose-Einstein condensation with evaporatively cooled atoms*. Contemp. Phys. **37**.1 (1996).

[GZ97] C. W. Gardiner, P. Zoller. *Quantum kinetic theory: A quantum kinetic master equation for condensation of a weakly interacting Bose gas without a trapping potential*. Phys. Rev. A **55** (4 1997).

[Sto97] H. T. C. Stoof. *Initial Stages of Bose-Einstein Condensation*. Phys. Rev. Lett. **78** (1997).

[WM97] D. Wolpert, W. Macready. *No free lunch theorems for optimization*. IEEE Trans. Evol. Comput. **1**.1 (1997).

[BGM98] J. Bastos de Figueiredo, C. Grotta Ragazzo, C. Malta. *Two important numbers in the Hénon-Heiles dynamics*. Phys. Lett. A **241**.1 (1998).

[GZ98] C. W. Gardiner, P. Zoller. *Quantum kinetic theory. III. Quantum kinetic master equation for strongly condensed trapped systems*. Phys. Rev. A **58** (1 1998).

[HS98] J. Harris, H. Stöcker. *Handbook of Mathematics and Computational Science*. Environmental Intelligence Uni. Springer New York, 1998.

[LLF98] I. Lagaris, A. Likas, D. Fotiadis. *Artificial neural networks for solving ordinary and partial differential equations*. IEEE Trans. Neural Netw. Learn. Syst. **9**.5 (1998).

[Mat+99] M. R. Matthews et al. *Vortices in a Bose-Einstein Condensate*. Phys. Rev. Lett. **83** (1999).

[OM99] D. Opitz, R. Maclin. *Popular Ensemble Methods: An Empirical Study*. J. Artif. Int. Res. **11**.1 (1999).

[Pin99] A. Pinkus. *Approximation theory of the MLP model in neural networks*. Acta Numer. **8** (1999).

[Sto99] H. T. C. Stoof. *Coherent Versus Incoherent Dynamics During Bose-Einstein Condensation in Atomic Gases*. J. Low Temp. Phys. **114**.1 (1999).

[ZNG99] E. Zaremba, T. Nikuni, A. Griffin. *Dynamics of trapped Bose gases at finite temperatures*. 1999. arXiv: `cond-mat/9903029 [cond-mat.stat-mech]`.

[Die00] T. G. Dietterich. *Ensemble Methods in Machine Learning*. *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000.

[GZ00] C. W. Gardiner, P. Zoller. *Quantum kinetic theory. V. Quantum kinetic master equation for mutual interaction of condensate and noncondensate*. Phys. Rev. A **61** (3 2000).

[HBG00] W. Hu, R. Barkana, A. Gruzinov. *Fuzzy Cold Dark Matter: The Wave Properties of Ultralight Particles*. Phys. Rev. Lett. **85** (2000).

[Mad+00] K. W. Madison, F. Chevy, W. Wohlleben, J. Dalibard. *Vortex Formation in a Stirred Bose-Einstein Condensate*. Phys. Rev. Lett. **84** (2000).

[Abo+01] J. R. Abo-Shaeer, C. Raman, J. M. Vogels, W. Ketterle. *Observation of Vortex Lattices in Bose-Einstein Condensates*. Science **292**.5516 (2001).

[DMB01] M. J. Davis, S. A. Morgan, K. Burnett. *Simulations of Bose Fields at Finite Temperature*. Phys. Rev. Lett. **87**.16 (2001).

[DS01] R. A. Duine, H. T. C. Stoof. *Stochastic dynamics of a trapped Bose-Einstein condensate*. Phys. Rev. A **65** (2001).

[Mod+01] G. Modugno et al. *Bose-Einstein condensation of potassium atoms by sympathetic cooling*. Science **294**.5545 (2001).

[SB01] H. T. C. Stoof, M. J. Bijlsma. *Dynamics of Fluctuating Bose–Einstein Condensates*. J. Low Temp. Phys. **124**.3 (2001).

[GAF02] C. W. Gardiner, J. R. Anglin, T. I. A. Fudge. *The stochastic Gross-Pitaevskii equation*. Journal of Physics B: Atomic, Molecular and Optical Physics **35**.6 (2002).

[Ket02] W. Ketterle. *Nobel lecture: When atoms behave as waves: Bose-Einstein condensation and the atom laser*. Rev. Mod. Phys. **74** (2002).

[DC03] C. M. Dion, E. Cancès. *Spectral method for the time-dependent Gross-Pitaevskii equation with a harmonic trap*. eng. Phys. Rev. E **67**.4 Pt 2 (2003).

[FW03] A. Fetter, J. Walecka. *Quantum Theory of Many-particle Systems*. Dover Books on Physics. Dover Publications, 2003.

[GD03] C. W. Gardiner, M. J. Davis. *The stochastic Gross–Pitaevskii equation: II*. J. Phys. B **36**.23 (2003).

[Mic+04] A. Micheli, A. J. Daley, D. Jaksch, P. Zoller. *Single Atom Transistor in a 1D Optical Lattice*. Phys. Rev. Lett. **93** (2004).

[MT04] G. N. Milstein, M. V. Tretyakov. *Stochastic Numerics for Mathematica Physics*. Springer Berlin Heidelberg, 2004.

[CPS06] K. Chellapilla, S. Puri, P. Simard. *High Performance Convolutional Neural Networks for Document Processing*. *Tenth International Workshop on Frontiers in Handwriting Recognition*. Université de Rennes 1. La Baule (France): Publisoft, 2006.

[Dim+07] S. Dimopoulos, P. W. Graham, J. M. Hogan, M. A. Kasevich. *Testing General Relativity with Atom Interferometry*. Phys. Rev. Lett. **98** (2007).

[Ryu+07] C. Ryu et al. *Observation of Persistent Flow of a Bose-Einstein Condensate in a Toroidal Trap*. Phys. Rev. Lett. **99** (26 2007).

[Sea+07] B. T. Seaman, M. Krämer, D. Z. Anderson, M. J. Holland. *Atomtronics: Ultracold-atom analogs of electronic devices*. Phys. Rev. A **75** (2007).

[Bla+08] P. B. Blakie et al. *Dynamics and statistical mechanics of ultra-cold Bose gases using c-field techniques*. Advances in Physics **57**.5 (2008).

[GW08] A. Griewank, A. Walther. *Evaluating Derivatives*. Second. Society for Industrial and Applied Mathematics, 2008.

[PJ08] N. P. Proukakis, B. Jackson. *PhD Tutorial: Finite-temperature models of Bose Einstein condensation*. Journal of Physics B Atomic Molecular Physics **41**.20, 203002 (2008). arXiv: 0810.0210 [cond-mat.other].

[Wei+08] C. N. Weiler et al. *Spontaneous vortices in the formation of Bose–Einstein condensates*. Nature **455**.7215 (2008).

[CSP09] A. D. Cronin, J. Schmiedmayer, D. E. Pritchard. *Optics and interferometry with atoms and molecules*. Rev. Mod. Phys. **81** (2009).

[Pep+09] R. A. Pepino, J. Cooper, D. Z. Anderson, M. J. Holland. *Atomtronic Circuits of Diodes and Transistors*. Phys. Rev. Lett. **103** (2009).

[Ste+09] S. Stellmer et al. *Bose-Einstein Condensation of Strontium*. Phys. Rev. Lett. **103** (2009).

[TCN09] M. Thalhammer, M. Caliari, C. Neuhauser. *High-order time-splitting Hermite and Fourier spectral methods*. J. Comput. Phys. **228**.3 (2009).

[UB09] R. Uetz, S. Behnke. *Large-scale object recognition with CUDA-accelerated hierarchical neural networks*. *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*. Vol. 1. 2009.

[Coc+10] S. P. Cockburn et al. *Matter-Wave Dark Solitons: Stochastic versus Analytical Results*. Phys. Rev. Lett. **104** (2010).

[GB10] X. Glorot, Y. Bengio. *Understanding the difficulty of training deep feedforward neural networks*. Vol. 9. Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS). Chia Laguna Resort, Sardinia, Italy, 2010.

[SKP10] D. Strigl, K. Kofler, S. Podlipnig. *Performance and Scalability of GPU-Based Convolutional Neural Networks*. *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. 2010.

[Cir+11] D. C. Cireşan et al. *Flexible, high performance convolutional neural networks for image classification*. *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*. IJCAI'11. Barcelona, Catalonia, Spain: AAAI Press, 2011.

[DHS11] J. Duchi, E. Hazan, Y. Singer. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. J. Mach. Learn. Res. **12**.61 (2011).

[HKW11] G. E. Hinton, A. Krizhevsky, S. D. Wang. *Transforming Auto-Encoders*. *Artificial Neural Networks and Machine Learning – ICANN 2011*. Ed. by T. Honkela, W. Duch, M. Girolami, S. Kaski. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[Lu+11] M. Lu, N. Q. Burdick, S. H. Youn, B. L. Lev. *Strongly Dipolar Bose-Einstein Condensate of Dysprosium*. Phys. Rev. Lett. **107** (2011).

[Ram+11] A. Ramanathan et al. *Superflow in a Toroidal Bose-Einstein Condensate: An Atom Circuit with a Tunable Weak Link*. Phys. Rev. Lett. **106** (13 2011).

[Aik+12] K. Aikawa et al. *Bose-Einstein Condensation of Erbium*. Phys. Rev. Lett. **108** (2012).

[Ben12] Y. Bengio. *Practical Recommendations for Gradient-Based Training of Deep Architectures*. Ed. by G. Montavon, G. B. Orr, K.-R. Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[CH12] P.-H. Chavanis, T. Harko. *Bose-Einstein condensate general relativistic stars*. Phys. Rev. D **86** (2012).

[KSH12] A. Krizhevsky, I. Sutskever, G. E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. *Adv. Neural Inf. Process. Syst. 32*. Ed. by F. Pereira, C. Burges, L. Bottou, K. Weinberger. Vol. 25. Curran Associates, Inc., 2012.

[Ris12] H. Risken. *The Fokker-Planck Equation*. 2nd ed. Springer Berlin Heidelberg, 2012.

[RBB12] S. J. Rooney, P. B. Blakie, A. S. Bradley. *Stochastic projected Gross-Pitaevskii equation*. Phys. Rev. A **86** (2012).

[Ber+13] T. Berrada et al. *Integrated Mach–Zehnder interferometer for Bose–Einstein condensates*. Nat. Commun. **4**.1 (2013).

[Hir13] A. Hirose. *Learning Algorithms in Complex-Valued Neural Networks using Wirtinger Calculus*. *Complex-Valued Neural Networks: Advances and Applications*. 2013.

[Mni+13] V. Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013.

[PDG13] N. P. Proukakis, M. J. Davis, S. A. Gardiner. *Selected Theoretical Comparisons for Bosons*. Imperial College Press, 2013. Chap. 17.

[Ryu+13] C. Ryu, P. W. Blackburn, A. A. Blinova, M. G. Boshier. *Experimental Realization of Josephson Junctions for an Atom SQUID*. Phys. Rev. Lett. **111**.20 (2013).

[Sut+13] I. Sutskever, J. Martens, G. Dahl, G. Hinton. *On the importance of initialization and momentum in deep learning*. *Proceedings of the 30th International Conference on Machine Learning*. Ed. by S. Dasgupta, D. McAllester. Vol. 28. Proc. Mach. Learn. Res. 3. Atlanta, Georgia, USA: PMLR, 2013.

[Wri+13] K. C. Wright et al. *Driving Phase Slips in a Superfluid Atom Circuit with a Rotating Weak Link*. Phys. Rev. Lett. **110** (2013).

[MK14] N. D. Matsakis, F. S. Klock II. *The rust language*. *ACM SIGAda Ada Letters*. Vol. 34. 3. ACM. 2014.

[SCB14] H.-Y. Schive, T. Chiueh, T. Broadhurst. *Cosmic structure as the quantum interference of a coherent dark wave*. Nature Physics **10**.7 (2014).

[Sri+14] N. Srivastava et al. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. J. Mach. Learn. Res. **15**.56 (2014).

[ASB15] M. Arjovsky, A. Shah, Y. Bengio. *Unitary Evolution Recurrent Neural Networks*. 2015.

[Gir15] R. Girshick. *Fast R-CNN*. 2015. arXiv: `1504.08083 [cs.CV]`.

[He+15] K. He, X. Zhang, S. Ren, J. Sun. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: `1502.01852 [cs.CV]`.

[MOS15] G. E. Marti, R. Olf, D. M. Stamper-Kurn. *Collective excitation interferometry with a toroidal Bose-Einstein condensate*. Phys. Rev. A **91** (2015).

[Mar+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015.

[Wil+15] K. E. Wilson, Z. L. Newman, J. D. Lowney, B. P. Anderson. *In situ imaging of vortices in Bose-Einstein condensates*. Phys. Rev. A **91** (2 2015).

[GBC16] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. MIT Press, 2016.

[LKT16] J. Ling, A. Kurzawski, J. Templeton. *Reynolds averaged turbulence modelling using deep neural networks with embedded invariance*. J. Fluid Mech. **807** (2016).

[PS16] L. Pitaevskii, S. Stringari. *Bose-Einstein Condensation and Superfluidity*. Oxford University Press, 2016.

[Sil+16] D. Silver et al. *Mastering the game of Go with deep neural networks and tree search*. Nature **529**.7587 (2016).

[Wig+16] P. B. Wigley et al. *Fast machine-learning online optimization of ultra-cold-atom experiments*. Sci. Rep. **6**.1 (2016).

[Bay+17] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind. *Automatic Differentiation in Machine Learning: A Survey*. J. Mach. Learn. Res. **18**.1 (2017).

[Kes+17] N. S. Keskar et al. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. 2017. arXiv: 1609.04836 [cs.LG].

[KB17] D. P. Kingma, J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[Lap17] A. Lapidoth. *A Foundation in Digital Communication*. 2nd ed. Cambridge University Press, 2017.

[Mem17] K. Y. Meme. *Вжух (Whoosh)*. 2017. URL: https://knowyourmeme.com/memes/whoosh-%D0%B2%D0%B6%D1%83%D1%85.

[SFH17] S. Sabour, N. Frosst, G. E. Hinton. *Dynamic Routing Between Capsules*. *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[Goy+18] P. Goyal et al. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. 2018. arXiv: 1706.02677 [cs.CV].

[HJE18] J. Han, A. Jentzen, W. E. *Solving high-dimensional partial differential equations using deep learning*. Proc. Natl. Acad. Sci. U.S.A. **115**.34 (2018).

[Hin18] G. Hinton. *Coursera: Neural Networks for Machine Learning (Lecture 6)*. 2018.

[HHS18] E. Hoffer, I. Hubara, D. Soudry. *Train longer, generalize better: closing the generalization gap in large batch training of neural networks*. 2018. arXiv: 1705.08741 [stat.ML].

[Inn18] M. Innes. *Flux: Elegant Machine Learning with Julia*. J. Open Source Softw. (2018).

[ML18] D. Masters, C. Luschi. *Revisiting Small Batch Training for Deep Neural Networks*. 2018.

[SS18] J. Sirignano, K. Spiliopoulos. *DGM: A deep learning algorithm for solving partial differential equations*. J. Comput. Phys. **375** (2018).

[Smi+18] S. L. Smith, P.-J. Kindermans, C. Ying, Q. V. Le. *Don't Decay the Learning Rate, Increase the Batch Size*. 2018. arXiv: 1711.00489 [cs.LG].

[ST18] I. Stewart, D. Tall. *Complex Analysis*. 2nd ed. Cambridge University Press, 2018.

[SB18] R. S. Sutton, A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[Tra+18] A. D. Tranter et al. *Multiparameter optimisation of a magneto-optical trap using deep learning*. Nat. Commun. **9**.1 (2018).

[Bro19] J. Brownlee. *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks*. 2019. URL: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/.

[GDY19] S. Greydanus, M. Dzamba, J. Yosinski. *Hamiltonian Neural Networks*. 2019. arXiv: 1906.01563 [cs.NE].

[LH19] I. Loshchilov, F. Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

[Nak+19] I. Nakamura et al. *Non-standard trajectories found by machine learning for evaporative cooling of 87Rb atoms*. Opt. Express **27**.15 (2019).

[Ope+19] OpenAI et al. *Learning Dexterous In-Hand Manipulation*. 2019. arXiv: 1808.00177 [cs.LG].

[Pas+19] A. Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. *Adv. Neural Inf. Process. Syst. 32*. Curran Associates, Inc., 2019.

[RPK19] M. Raissi, P. Perdikaris, G. Karniadakis. *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*. J. Comput. Phys. **378** (2019).

[Bar+20] A. J. Barker et al. *Applying machine learning optimization methods to the production of a quantum gas*. Mach. Learn.: Sci. Technol. **1**.1 (2020).

[Cra+20] M. Cranmer et al. *Lagrangian Neural Networks*. 2020. arXiv: 2003.04630 [cs.LG].

[Dav+20] E. T. Davletov et al. *Machine learning for achieving Bose-Einstein condensation of thulium atoms*. Phys. Rev. A **102** (2020).

[Fla20] P. Flannigan. *Charlie*. 2020. URL: https://m.facebook.com/PaulFlanniganPhotography/photos/a.287249921423539/1754046811410502/?type=3.

[Ha+20] S. Ha et al. *Learning to Walk in the Real World with Minimal Human Effort*. 2020. arXiv: 2002.08550 [cs.RO].

[Mis20] D. Misra. *Mish: A Self Regularized Non-Monotonic Activation Function*. 2020. arXiv: 1908.08681 [cs.LG].

[Nes+20] G. Ness et al. *Single-Exposure Absorption Imaging of Ultracold Atoms Using Deep Learning*. Phys. Rev. A **14**.1 (2020).

[RSB20] C. Ryu, E. C. Samson, M. G. Boshier. *Quantum interference of currents in an atomtronic SQUID*. Nat. Commun. **11**.1 (2020).

[Xu+20] Y. Xu et al. *Solving Fokker–Planck equation using deep learning*. Chaos **30**.1 (2020).

[Ami+21] L. Amico et al. *Roadmap on Atomtronics: State of the art and perspective*. AVS Quantum Sci. **3**.3 (2021).

[Guo+21] S. Guo et al. *Machine-learning enhanced dark soliton detection in Bose–Einstein condensates*. Mach. Learn.: Sci. Technol. **2**.3 (2021).

[Kar+21] G. E. Karniadakis et al. *Physics-informed machine learning*. Nat. Rev. Phys. **3**.6 (2021).

[Lu+21] L. Lu, X. Meng, Z. Mao, G. E. Karniadakis. *DeepXDE: A Deep Learning Library for Solving Differential Equations*. SIAM Rev. **63**.1 (2021).

[Met+21] F. Metz, J. Polo, N. Weber, T. Busch. *Deep-learning-based quantum vortex detection in atomic Bose–Einstein condensates*. Mach. Learn.: Sci. Technol. **2**.3 (2021).

[Cha+22] H. Y. J. Chan et al. *The diversity of core–halo structure in the fuzzy dark matter model*. Mon. Not. R. Astron. Soc. **511**.1 (2022).

[Cuo+22] S. Cuomo et al. *Scientific Machine Learning through Physics-Informed Neural Networks: Where we are and What's next*. 2022. arXiv: 2201.05624 [cs.LG].

[Kim+22] M. Kim et al. *Suppression of spontaneous defect formation in inhomogeneous Bose gases*. Phys. Rev. A **106** (2022).

[Ven+22] Z. Vendeiro et al. *Machine-learning-accelerated Bose-Einstein condensation*. Phys. Rev. Res. **4** (2022).

[HG23] D. Hendrycks, K. Gimpel. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: 1606.08415 [cs.LG].

[Kee+23] N. Keepfer, T. Flynn, N. Parker, T. Billam. *SuperVortexNet: Reconstructing Superfluid Vortex Filaments Using Deep Learning*. 2023.

[Mil+23] N. Milson et al. *High-dimensional reinforcement learning for optimization and control of ultracold quantum gases*. Mach. Learn.: Sci. Technol. **4**.4 (2023).

[Rab+23] T. Rabga et al. *Variations of the Kibble-Zurek scaling exponents of trapped Bose gases*. Phys. Rev. Lett. **108**.2 (2023).

[WY23] R. Wang, R. Yu. *Physics-Guided Deep Learning for Dynamical Systems: A Survey*. 2023. arXiv: 2107.01272 [cs.LG].

[App24] Apple. *Apple Developer — Machine Learning*. 2024. URL: https://developer.apple.com/machine-learning/ (visited on 2023).

[Gri24a] J. Griffiths. *Complex-Valued Neural Initial Value Problems codebase*. 2024. URL: https://github.com/0jg/ml-ivp-complex (visited on 2024).

[Gri24b] J. Griffiths. *Machine learning thermodynamical parameters of Bose gases*. 2024. URL: https://github.com/0jg/mu_T_prediction (visited on 2024).

[Gri24c] J. Griffiths. *Neural Initial Value Problems codebase*. 2024. URL: https://github.com/0jg/ml-ivp (visited on 2024).

[Gri24d] J. Griffiths. *Stochastic Gross-Pitaevskii Equation in Rust*. 2024. URL: https://github.com/0jg/sgpe-rs.

[GWG24] J. Griffiths, S. A. Wrathmall, S. A. Gardiner. *Solving physics-based initial value problems with unsupervised machine learning*. 2024. arXiv: 2407.18320 [physics.comp-ph].

[Ind+24] M. Indjin, I.-K. Liu, N. P. Proukakis, G. Rigopoulos. *Virialized profiles and oscillations of self-interacting fuzzy dark matter solitons*. Phys. Rev. D **109** (2024).

[PyT24] PyTorch. *torch.nn.Conv2d documentation*. 2024. URL: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html.

[Sta24] Stack Overflow. *Stack Overflow 2024 Developer Survey*. 2024. URL: https://survey.stackoverflow.co/2024/.

[Ten24] Tensorflow. *tf.keras.layers.Conv2D*. 2024. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D.